Adaptive Resource Location in a Peer-to-Peer Network

By
Michael Iles

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

June 2002

© Copyright
2002, Michael Iles

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-79757-0

Canada

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis,

Adaptive Resource Location in a Peer-to-Peer Network

submitted by

Michael Iles

(Director, School of Computer Science)

(Thesis Supervisor)

Carleton University
June 2002

Abstract

In this thesis we develop a ‘flooding broadcast meta-protocol’ which is able to describe a broad range of flooding broadcast network protocols, including the Gnutella, s-peer and FastTrack/KaZaA protocols and a subset of the JXTA protocol. We then employ genetic programming to derive network protocols from this meta-protocol that are optimal in terms of resource-discovery for various network scenarios. In all cases the evolved protocols meet or exceed the resource-discovery performance of the Gnutella protocol, confirming the viability of this approach. The resulting optimal strategies are analyzed and catalogued according to their behaviour, and insights into efficient resource-discovery techniques in peer-to-peer networks are presented.

Acknowledgements

I would like to thank my advisor, Dwight Deugo, for inspiring and informing me; my family and friends, for tolerating my periods of absence; and Julie, for her love.

Table of Contents

Abstract	iii
Acknowledgements	iv
Table of Contents	v
List of Tables	vii
List of Figures	viii
List of Appendices	xi
1 Introduction	1
1.1 Context and Motivation	2
1.2 Problem	3
1.3 Goal	4
1.4 Expected Contributions	4
1.5 Outline	5
2 Background	6
2.1 Small World Networks	7
2.2 Resource Discovery	9
2.2.1 Centralized Indices	9
2.2.2 Replicated Indices	13
2.2.3 Flooding Broadcast Networks	14
2.2.4 Distributed Hashing Systems	27
2.2.5 Alternatives	40
2.2.6 Resource Discovery Summary	42
2.3 Genetic Programming	43

2.3.1	The Process	43
2.3.2	Genetic Algorithms	46
2.3.3	Benefits	47
2.3.4	Successes	47
2.4	Conclusion	48
3	Approach	50
3.1	The Simulation	51
3.2	The Flooding Broadcast Meta-Protocol	55
3.3	Genetic Programming	60
3.4	Summary	62
4	Results	64
4.1	The Scenarios	64
4.2	Presentation of Results	67
4.3	Results	70
4.4	Performance	115
4.5	Conclusion	115
5	Analysis	116
5.1	Scenarios A and B	116
5.1.1	Gnutella	116
5.1.2	FBMP	118
5.2	Scenario C	122
5.2.1	Gnutella	122
5.2.2	FBMP	123
5.3	Scenarios D, E and F	124
5.4	Summary	125
5.5	Conclusion	127
6	Conclusion	128
6.1	Confirmation of Approach	128
6.2	Contributions	130
6.3	Future Work	131
	Appendices	134
	Bibliography	139

List of Tables

2.1	Comparison of Resource Discovery Techniques	42
3.1	CONN Expression Terminals	57
3.2	RANK Expression Terminals	58
4.1	Breakdown of Scenarios	66
B.1	Genetic programming parameters	138
B.2	Peer-to-peer simulation parameters	138

List of Figures

2.1	A broadcast message from s and a response by r	19
2.2	The Crossover Operation	45
2.3	The Mutation Operation	45
3.1	Simulated physical network	53
3.2	Simulated logical network	53
3.3	The 7-neighbour Gnutella protocol (left) and the s-peer protocol as expression trees.	60
3.4	Fitness Function	61
4.1	Scenario A-1 evolved network topology	71
4.2	Scenario A-1 evolved protocol expressions	71
4.3	Scenario A-1 performance graphs.	72
4.4	Scenario A-2 evolved network topology	73
4.5	Scenario A-2 evolved protocol expressions	73
4.6	Scenario A-2 performance graphs.	74
4.7	Scenario A-3 evolved network topology	75
4.8	Scenario A-3 evolved protocol expressions	75
4.9	Scenario A-3 performance graphs	76
4.10	Scenario A-4 evolved network topology	77
4.11	Scenario A-4 evolved protocol expressions	77
4.12	Scenario A-4 performance graphs	78
4.13	Scenario A-5 evolved network topology	79

4.14	Scenario A-5 evolved protocol expressions	79
4.15	Scenario A-5 performance graphs	80
4.16	Scenario A-6 evolved network topology	81
4.17	Scenario A-6 evolved protocol expressions	81
4.18	Scenario A-6 performance graphs	82
4.19	Scenario A-7 evolved network topology	83
4.20	Scenario A-7 evolved protocol expressions	83
4.21	Scenario A-7 performance graphs	84
4.22	Scenario B-1 evolved network topology	85
4.23	Scenario B-1 evolved protocol expressions	85
4.24	Scenario B-1 performance graphs	86
4.25	Scenario B-2 evolved network topology	87
4.26	Scenario B-2 evolved protocol expressions	87
4.27	Scenario B-2 performance graphs	88
4.28	Scenario B-3 evolved network topology	89
4.29	Scenario B-3 evolved protocol expressions	89
4.30	Scenario B-3 performance graphs	90
4.31	Scenario B-4 evolved network topology	91
4.32	Scenario B-4 evolved protocol expressions	91
4.33	Scenario B-4 performance graphs	92
4.34	Scenario B-5 evolved network topology	93
4.35	Scenario B-5 evolved protocol expressions	93
4.36	Scenario B-5 performance graphs	94
4.37	Scenario B-6 evolved network topology	95
4.38	Scenario B-6 evolved protocol expressions	95
4.39	Scenario B-6 performance graphs	96
4.40	Scenario B-7 evolved network topology	97
4.41	Scenario B-7 evolved protocol expressions	97
4.42	Scenario B-7 performance graphs	98

4.43	Scenario C-1 evolved network topology	99
4.44	Scenario C-1 evolved protocol expressions	99
4.45	Scenario C-1 performance graphs	100
4.46	Scenario C-2 evolved network topology	101
4.47	Scenario C-2 evolved protocol expressions	101
4.48	Scenario C-2 performance graphs	102
4.49	Scenario C-3 evolved network topology	103
4.50	Scenario C-3 evolved protocol expressions	103
4.51	Scenario C-3 performance graphs	104
4.52	Scenario C-4 evolved network topology	105
4.53	Scenario C-4 evolved protocol expressions	105
4.54	Scenario C-4 performance graphs	106
4.55	Scenario C-5 evolved network topology	107
4.56	Scenario C-5 evolved protocol expressions	107
4.57	Scenario C-5 performance graphs	108
4.58	Scenario D evolved network topology	109
4.59	Scenario D evolved protocol expressions	109
4.60	Scenario D performance graphs	110
4.61	Scenario E evolved network topology	111
4.62	Scenario E evolved protocol expressions	111
4.63	Scenario E performance graphs	112
4.64	Scenario F evolved network topology	113
4.65	Scenario F evolved protocol expressions	113
4.66	Scenario F performance graphs	114
5.1	Average node degrees of evolved FBMP networks for Scenarios A and B	119
A.1	The offline bandwidth-calculation algorithm	136

List of Appendices

A	Bandwidth Algorithms	134
A.1	Offline Algorithm	135
B	Simulation Parameters	137

Chapter 1

Introduction

The Internet has carried us into an era of tremendous connectivity, one in which reliable and efficient communication is possible with all corners of the globe. The information available to us increases daily, but the tools we use to find that information are struggling to cope.

The decentralized network—a network in which machines interact with each other as relative equals instead of in a master and slave relationship—was a common paradigm in the early days of networking, as the widely-distributed Internet took form to link pre-existing computing centres on equal footings. Usenet was built as a decentralized news-dissemination network with the ‘Unix to Unix Copy Protocol’ (UUCP), and the Internet’s routing was based on the Domain Name System (DNS), a decentralized index mapping machine names to numerical addresses. As the Internet evolved and traffic rose, however, a fairly static web of dedicated communication servers took over the job of routing packets and the majority of machines were pushed to the fringes. The underlying communications infrastructure remains, but increasing connection speeds and the rising power of desktop machines has brought a resurgence of interest in decentralized, or peer-to-peer networks.

Whether this interest stems from popular dissatisfaction with centralized publishing mechanisms, people's desire to trade (or steal) software and music, or some other purely technical merit of decentralized networks, the interest in and utility of peer-to-peer networks as mechanisms for resource discovery and content distribution cannot be denied.

1.1 Context and Motivation

Napster's brief reign provided a glimpse of the potential of networks that encourage 'content from the people'. Estimates of the content available at Napster's peak range well into the terabytes, achieved with relatively little centralized administration or costly hardware. Ultimately however, Napster's centralized architecture proved to be its downfall as the company maintaining the servers was unable to defend itself from legal attacks.

Decentralized networks that serve the same function have risen in Napster's place though, and peer-to-peer networks in general have recently drawn tremendous interest from both the academic communities and the general public. The potential for such networks is great: fault tolerance, storage, content distribution, anonymity, public naming mechanisms, individual publishing, distributed cost, censorship-resistance, and resource discovery are some of the abilities touted.

Among the many extant and proposed types of peer-to-peer networks, the broad category of *flooding broadcast networks* has generated by far the most popular interest. The combined size of the flooding broadcast Gnutella and FastTrack (KaZaA) networks ranges into the millions of peers. There are other viable peer-to-peer alternatives—most notably the distributed hashing systems, discussed in Chapter 2,

which have their own set of drawbacks—but flooding broadcast networks have proved their utility and robustness in many real-world scenarios.

1.2 Problem

Flooding broadcast networks have proven themselves capable of meeting most major criteria for successful peer-to-peer networks (principally decentralized administration and robustness to peer transience and failure), but they have one major shortcoming: an inefficient resource discovery mechanism that has difficulty scaling.

Employing a social analogy, the problem is this: given a typical network of friends and acquaintances and some desired information such as leads on apartments to rent, what approach should you employ to maximize the number of apartment leads that your friends inform you of? Further, imagine that your communications are restricted to a small number of telephone calls per day. What if some of your friends are always looking for two-bedroom apartments, while others are always looking for three bedrooms? What if one friend never responds to your requests? What if another calls constantly and uses up your precious phone calls?

There is no obvious solution in the social analogy, and indeed the solution most likely depends on the specifics of the housing market and the social network in question. The same is true of the corresponding decentralized resource discovery problem: the number of neighbours a peer should maintain, how it should choose those neighbours, how it should keep from becoming overloaded with traffic, and how varying bandwidths and search scopes should be accommodated are all open for debate. A

wide range of protocols ¹ to achieve resource discovery in flooding broadcast peer-to-peer networks have sprung up, with no clear winner among them.

1.3 Goal

All of the flooding broadcast protocols currently in use have one attribute in common: they are hand-crafted heuristics. In other words, they are a human designer’s best guess at how peers in a decentralized network should interact.

In this thesis we propose the use of genetic programming, a machine learning technique, to remove some of the human bias² from this search for heuristics. Our goal is to show that this approach is capable of producing optimal flooding broadcast network protocols for arbitrary network scenarios.

1.4 Expected Contributions

Our expected contributions are twofold. First, we aim to develop a ‘meta-protocol’ for flooding broadcast networks that is expressive enough to describe a range of existing and deployed flooding broadcast network protocols, and then to demonstrate that genetic programming is a viable method to obtain specific network protocols from this meta-protocol that are optimized for various specific network scenarios.

Our second expected contribution is an analysis of these new heuristic protocols

¹The term *protocol* is used in this thesis not in the narrow computer science sense of a wire protocol, but in the broader sense of an earthquake protocol or diplomatic protocol: a set of rules to be followed in specific situations.

²We recognize that human bias is unavoidable in the design and execution of any experiment, and in the specific case of genetic programming human bias is manifested most prominently in the designer’s choice of fitness function. Recognizing this, we still claim that the empirical optimum achieved by an evolutionary search is more free of human bias than a hand-crafted heuristic and can be justified on the basis of its numerical superiority over a wide range of competing solutions.

to provide insight into optimal resource discovery strategies in various network scenarios. Insights gained here will be of benefit to any peer-to-peer protocol designer, independent of the flooding broadcast meta-protocol or the genetic programming arrangement used to arrive at them.

1.5 Outline

Chapter 2 discusses other techniques that have been employed to solve the resource discovery problem in both centralized and decentralized networks, and introduces the concepts of genetic programming and ‘small world’ networks. Chapter 3 develops the general flooding broadcast network protocol and describes the network model and genetic programming mechanisms employed in the simulated peer-to-peer network.

Chapter 4 presents the experimental results obtained from the simulation mentioned above. Chapter 5 provides an analysis of these results and draws out some new insights into optimal strategies for resource discovery in flooding broadcast networks. Chapter 6 evaluates the success achieved in meeting the goals of the thesis, outlines the contributions made and discusses future work.

Chapter 2

Background

There has been a tremendous recent interest in peer-to-peer networks, and consequently a large amount of work has been produced. This work has involved the Web and wireless networks, but has primarily focused on *overlay networks*¹ superimposed onto the Internet. Using the Internet as a communications substrate means that the overlay networks are constrained by the same factors constraining the majority of machines on the Internet: relatively low bandwidths and limited processing power. Section 2.2 surveys this burgeoning field and discusses the most interesting resource discovery strategies emerging from it.

Before delving into peer-to-peer systems however, Section 2.1 introduces the study of ‘small world’ networks, a concept that will help in evaluating the approaches of some of the peer-to-peer systems examined.

The chapter concludes with an introduction to genetic programming, the evolutionary search technique that will allow us to explore the design space of the general flooding broadcast network protocol we will develop in Chapter 3.

¹An overlay network is a virtual network superimposed onto a lower-level network and using that lower-level network to provide communication services.

2.1 Small World Networks

A ‘small-world’ network is one which exists somewhere between a completely regular lattice and a completely random network. Both regular and random networks have been studied extensively, but Watts and Strogatz have pointed out [WS98] that neither extreme accurately represents many networks observed in the real world.

Watts and Strogatz began their research into this area by defining two measurements of a network: the *clustering coefficient* and the *characteristic path length*. Loosely, the clustering coefficient is the degree to which the network shows local clustering or ‘cliquishness’. It was initially defined as the average of $\gamma_v = \frac{|E(\Gamma_v)|}{\binom{k_v}{2}}$ over all vertices where $|E(\Gamma_v)|$ is the number of edges in the neighbourhood of vertex v and k_v is the degree of v . This definition is unclear about the treatment of degree-0 and degree-1 vertices however, and the authors have since adopted the definition $C = \frac{3 \times \text{number of triangles on the graph}}{\text{number of connected triples of vertices}}$ [NWS02]. (This thesis adopts the latter definition.) The characteristic path length of a graph is the average shortest length between any two vertices in the graph.

Using these measurements, Watts and Strogatz observed that regular lattices are both highly clustered and have a high characteristic path length, while random networks are unclustered and have low characteristic path lengths. The transition from ordered to unordered in their model is achieved by ‘rewiring’ connections in the regular network. The initial rewiring method involved moving edges, and had a non-zero probability of leading to a disconnected network and consequently an infinite characteristic path length. This was later remedied with another method that added but did not remove edges [NW]. Their key observation was that during a network’s transition from a regular lattice to a random network, the characteristic path length

and clustering coefficient do not vary together. To underscore this, they applied the ‘small-world’ label to networks occupying the intermediate zone of high clustering coefficients and low characteristic path lengths.

Some initial motivation for this line of research came from an experiment done by Stanley Milgram in the late 1960s [Mil67]. Milgram delivered a number of letters to people in Nebraska which were addressed only by name to a stockbroker in Boston, Massachusetts. His instructions were that the recipients should pass the letters on to anyone they were on a first-name basis with and who they thought might be better able to guide the letters to their specified destination. Many of the letters were successfully delivered to the stockbroker, and Milgram found that the number of intermediate friends involved varied between two and ten but averaged five. This result has since passed into common use as the idea of ‘six degrees of separation’ between any two people in the world. These types of social networks—cliquish and yet well-connected—were what Watts and Strogatz were seeking to represent.

The high clustering and low characteristic path length of small-world networks is what makes them an interesting analog to many real-world networks: Watts and Strogatz initially characterized the neural network of the *Caenorhabditis Elegans* worm, the transformer map of the Western United States’ electrical grid and the collaboration graph of film actors as small-world networks. They later found that it was easier for a cellular automata to perform a task called ‘density classification’ on a small-world graph than on a regular lattice; they found that in iterated multi-player games of the prisoners’ dilemma co-operation arose less frequently on a small-world graph than on a regular lattice; they also found that a small-world topology helped oscillator networks to synchronize more easily than on a regular lattice [New00] [Wat99].

In the context of peer-to-peer networks, it has been shown that the Gnutella network (introduced in Section 2.2.3) exhibits strong small-world characteristics [Jov00]. The small-world concept and measurements introduced by Watts and Strogatz will aid us in discussing the peer-to-peer approaches in the following section and in analyzing the results presented in Chapter 4.

2.2 Resource Discovery

Resource discovery remains one of the most difficult problems facing the designers of peer-to-peer systems. Andy Oram writes in a recent book on peer-to-peer systems [Ora01] that no-one among the authors considered felt qualified to write on the topic of resource discovery, and that he feels that this is indicative of the low level of sophistication the field has so far achieved.

This section broadly categorizes a number of the most important peer-to-peer systems according to their method of resource discovery. These approaches range from centralized and decentralized indices through flooding broadcast networks, distributed hashing systems, and on to some alternative approaches and hybrids.

2.2.1 Centralized Indices

The simplest way to facilitate resource discovery in a network is to maintain a centralized index of all available resources. Assuming that the central index resides on a host with sufficient processing power and bandwidth, this arrangement is highly efficient because only one request and one reply are required to effectively query the entire network.

The drawbacks of this arrangement are significant, though: providing a central

host with sufficient processing power and bandwidth is expensive, and having the index available from only one host creates a single point of failure for the network – in both a physical and a legal sense. Scalability is an issue because the only way to increase the size of the network is to add processing power and bandwidth to the central index’s host. As well, the range of queries that can be made of the index is limited by the information that the index actually stores – if the data doesn’t exist in the index, the client cannot query for it.

Any client-server arrangement is an example of a central index, but the following examples of central indices exist in the domain of widely distributed, low-bandwidth overlay networks that this thesis concentrates on.

A client running **Napster** [Nap] connects to a globally-known central server and transfers to it the list of files the client is willing to share. When the client queries the central server for a file, the server returns the addresses of other clients who have offered to share that file and the two clients proceed to negotiate the transfer without any further central involvement. Thus the client can effectively search the entire catalog of all other clients with one query and one reply, contingent on the availability of the central server hosting the index.

Napster was released in May of 1999, and by December of that year it had drawn the ire of the Recording Industry Association of America in the form of a lawsuit. It mounted various legal responses and was pressed to offer concessions including screening its index for copyrighted material, but was eventually shut down. Many imitators such as OpenNap, Aimster and CuteMX have attempted to play the same role, but all are in various degrees of legal trouble.

The rise and fall of Napster is instructive for file-sharing applications of the future.

The American legal system has made it clear that a legal entity hosting a system that exists largely to facilitate illegal activity (copyright violations, in this case) is itself liable for that activity. This distinction continues to be vague in other areas such as holding Internet Service Providers (ISPs) responsible for their customers' downloading of illegal materials – in this case, the service provided by the ISPs does not exist largely to facilitate such illegal actions, and requiring ISPs to monitor all traffic through their computers in such a case is not necessary or even viable. The point is that maintaining a central index will in most cases make the index maintainer responsible for the legality of its contents. Napster fell because its centralized index was a single point of failure vulnerable to a legal attack.

Though the objectives of file-sharing applications like Napster may be morally suspect, there is no doubt that the broader goal of information sharing in the face of possible corporate or political censorship remains a worthy objective.

Another way to achieve a centralized index is through the use of a hierarchy of centralized servers. The **Secure Discovery Service** [CZH⁺99] uses a hierarchical arrangement of servers to cache device capabilities for network-enabled mobile devices and to handle requests for those devices.

A **Web search engine** such as Google or AltaVista is another form of centralized index. These indices are maintained by Web crawlers which actively discover new pages to index by following links from known pages or by acting on information submitted by third parties. Web search engines share the same attributes as other centralized indices: highly efficient, expensive, and physically and legally vulnerable.

Web search engines provide an interesting example of another aspect of resource location in a network: trust. Web search engines rank the returned results as they

see fit, which can involve arrangements in which resource providers pay the search engine maintainer to have their resources elevated in the listings. Similar problems may also arise without the knowledge of the search engine if resource providers, for example, bombard a search engine with resource submissions in an attempt to trick the engine into ranking the resources more highly. Others have attempted to influence search engines' rankings by exploiting details of the engines' ranking mechanisms. For example, the Church of Scientology has been shown [pts02] to have exploited Google's page ranking mechanisms by maintaining a number of seemingly independent sites which linked heavily to each other. Google interpreted this as evidence of the importance of the pages, and elevated their rankings.

No resource-location mechanism described in this chapter deals adequately with dishonest or manipulative participants in a network, but centralized indices, being under autocratic control and providing a single target for manipulation, are particularly susceptible.

Another example of a spidering approach to resource discovery is **OpenCOLA**, which maintains 'folders' that utilize agents to co-operatively search the network for resources based on user-defined criteria.

In this section we have seen that the primary benefit of a centralized index is that it is conceptually simple and highly efficient, but in a context that requires resilience in the face of censorship or legal prosecution, the central index's vulnerability is a critical weakness.

2.2.2 Replicated Indices

Maintaining a centralized index requires a significant investment in hardware and bandwidth by one party, and provides a single point of failure for the network. Replicating the index on multiple hosts in the network improves the resilience of the network by adding redundancy, and shares the query load across all the hosts thus allowing the network to scale more easily. The obvious drawback is that more control traffic is required to keep the replicated indices synchronized, and that the replication opens the possibility of out-of-date data.

The **Wide Area Network Service Location** (WANSL) service [RSS97] is one example of a system that uses a distributed index. WANSL is a wide-area extension of RFC 2165, a ‘Service Location Protocol’ designed to facilitate the discovery of network services. WANSL uses a replicated database of mappings to allow mobile devices to discover resources in a widely-distributed network that spans administrative domains.

Giggle [FIR⁺02] is a framework which uses distributed indices to create ‘GIGa scale Global Location Engines’ which are primarily intended to locate replicas of a resource in a network.

The Internet’s **Domain Name System** (DNS) [MD88] provides name-to-IP-address mapping for the global network. which it achieves by replicating the index across all DNS servers. It is remarkable that DNS has scaled virtually unchanged from its initial form to currently serve many millions of requests per second across the entire Internet. This scalability depends on the relatively static nature of globally-recognized domain names, the low churn rate of DNS hosts, the strict hierarchy that domain names are organized into and the presence of a global naming authority to authorize changes. One price that is paid for this scalability and speed is that DNS

changes typically take on the order of days to propagate to all servers.

A final example of a replicated index is the **Lightweight Directory Access Protocol** (LDAP). LDAP was designed as a minimal interface to an X.500 directory, and provides a facility to replicate directories across multiple directory servers, effectively achieving a distributed index.

We see from these examples that a distributed index mitigates some of the problems of a centralized index. A distributed index is more fault-tolerant and provides some load-sharing. It requires more co-ordination to maintain coherency across multiple hosts however, and opens the possibility of out-of-date data.

2.2.3 Flooding Broadcast Networks

An entirely different technique for performing resource location in a distributed network is to eliminate the idea of an index altogether and to directly query every node in the network. While this has obvious implications for performance and scalability, it can lead to a highly flexible and robust network.

‘Flooding broadcast’ refers to a ‘gossip’ or ‘epidemic’ style of message propagation in a forwarding network.² In these networks queries for a resource are propagated outwards in a breadth-first fashion from their source until either every peer has received it or the propagation reaches a predetermined horizon.

The explosion of interest in file-sharing networks has focused a lot of attention recently on flooding broadcast networks such as Gnutella and KaZaA. These networks have been criticized on numerous fronts, but the two biggest advantages they possess are unassailable: robustness and simplicity.

²A forwarding network is a peer-to-peer network in which peers co-operate by forwarding messages from one neighbour to another according to some routing scheme.

Robustness in the face of physical failures is always advantageous, but given the intense legal pressures applied to file-sharing networks, robustness in the face of legal prosecution is critical. With their lack of any centralized control, forwarding networks offer no legal target to attack. This robustness is also beneficial in the typical context that file-sharing networks occur in: a widely-distributed network involving highly-transient peers.

Though robustness is the critical trait, simplicity is the other factor contributing to the interest in flooding broadcast networks. There are other networks (such as the distributed hashing systems described in Section 2.2.4) that are decentralized but don't suffer from the same performance constraints that flooding broadcast networks do, but their workings are complicated enough to hinder their widespread adoption. A peer participating in a flooding broadcast network must hew to only a few very simple rules. In the case of Gnutella, the simplicity of the protocol has allowed many different clients to arise and to interact within the same network.

A secondary benefit of flooding broadcast networks that arises from the elimination of the shared index is that each peer can interpret any requests it receives in whatever manner it sees fit. This allows for infinite flexibility in the way that querying is done, ranging from simple name or full-text searches to queries that consist of mobile code running on the peer being queried.

These benefits come at a price, however, and flooding broadcast networks have many downsides. In a pure flooding broadcast network in which every peer receives every query, the bandwidth required grows exponentially with a linear increase in the number of peers. And because every peer needs to process the query, the time required is significantly more than that required to query one or a small number of

central hosts. This form of searching also doesn't have a distinct endpoint – as the search propagates outwards, there is no way for the querying peer to know whether to continue waiting for a response or whether all the other peers have received and processed the query.

Decentralized networks are also vulnerable to fragmentation. This could result from true fragmentation in which one group of peers becomes physically separated from another, or it could be 'effective' fragmentation if all peers along a border between two distinct groups become overwhelmed with traffic and cease to participate in the network.

Finally, forwarding networks are vulnerable to many types of malicious activity: denial of service attacks by sending out bogus queries, eavesdropping on traffic from other peers, forgery of search requests or search results, maliciously modifying, mis-routing or dropping others' traffic, and returning spam or other advertising disguised as search results. In a general sense it is possible to detect malicious activity in a forwarding network, but this requires explicit support from the protocol. The Gnutella protocol does not provide such support, and so detecting malicious activity is difficult: in many cases, there is no way to know which peer is tampering with the network traffic, or even to know whether traffic is being tampered with at all. In the case of a peer forwarding an inordinate number of requests, it would be difficult to distinguish a denial-of-service attack from a legitimately large set of queries. The issue of trust and vulnerability in peer-to-peer networks is a significant one, and no approach described in this chapter treats it adequately.

A common enhancement to a pure flooding broadcast network is the introduction of 'supernodes' – peers which have above-average resources and bandwidth and can

function as servers for the rest of the network. With supernodes peers can forward requests only to their neighbouring supernodes and the supernodes can propagate queries only amongst themselves, which decreases the total amount of traffic required to query the whole network.

Supernodes alleviate some of the performance concerns with flooding broadcast networks, but come with another set of restrictions. They increase the complexity of the network protocol, raising the bar for independent peering implementations to join the network. More importantly though, they rely on the asymmetric donation of resources and bandwidth to the network: supernodes contribute their resources to enable other more constrained peers to participate in the network.

Another common enhancement is to implement some form of caching to speed resource discovery. These enhancements range from simple least-recently-used caches to independently-maintained tries (radix trees) that index all observed content on the network [FV02].

Gnutella is probably the canonical flooding broadcast network. It was the earliest on the scene in the wake of Napster, and has been widely studied and critiqued since. It is—in its simplest incarnation—a pure, fully-decentralized flooding broadcast network.

The original Gnutella peer was released in early 2000 by Nullsoft employees who had been newly bought-out by AOL and were given free reign to pursue projects they considered interesting. AOL quickly realized that the use of Gnutella would conflict with its own goals and retracted the application, but the protocol was reverse-engineered and widely implemented. These independent implementations are flourishing today, and include BearShare, Bodetella, FileNavigator, Furi (now Phex),

Gnewtella, Gnewtellium, Gnotella, gnucleus, Gnut, gtk-Gnutella, LimeWire, Mactella, Morpheus (ex- of FastTrack), Mutella, N-tella, NapShare, Qtella, Swapnut, Swapper, ToadNode and Xolox.

The Gnutella protocol is a simple flooding broadcast protocol. A peer joins the network by obtaining the address of any other peer already in the network, and then broadcasts an advertisement of its presence through that peer to the rest of the network. Peers receiving this advertisement respond in turn with their own addresses.

A peer re-broadcasts messages it receives from one neighbour to the rest of its neighbours, so broadcasts propagate outwards in breadth-first fashion subject to a maximum depth. This depth, or *time-to-live*, effectively imposes a horizon beyond which a message will not propagate. Each peer maintains a record of all broadcast messages it has received (or a sliding window of the most recent), and the neighbour that the message was received from. When another peer responds to a request, that response is routed back to the original sender based on each intermediate peer's memory of the path the original broadcast message took³. This broadcast and reply behaviour is illustrated in Figure 2.1, which shows the outward propagation of a broadcast message and the reverse path traced by a response.

The Gnutella protocol has remained essentially unchanged since its inception, but the ease of implementing a peer has fostered a number of modifications to the peering behaviour. Many peers now attempt to ascertain some details about the machine they are running on and its connection to the Internet and to tailor the peer's participation in the network to its available resources. Typical behaviour is to favour

³A respondent could theoretically bypass the chain of nodes between itself and the originator and respond directly to the originating node, but this is not viable in practice because of firewalls and the prohibitive overhead of creating a new TCP socket solely for this purpose.

Figure 2.1: A broadcast message from s and a response by r . Only the connections that the message actually propagated on are shown; there could be additional connections between peers, but any subsequent copies of messages received by a peer are ignored.

high-bandwidth connections and to adjust the number of connections proportionally with the available bandwidth, which has the effect of creating a core subnetwork of high-performance hosts with the slower clients arranged around the periphery.

A study published in late 2000 [Cli00] showed that the Gnutella network had grown to the point where the average amount of traffic processed by a peer would saturate a dialup connection, rendering that peer unable to participate in the network. This led to the effective fracturing of the network into disjoint subnetworks bordered by the swamped peers and to the development of the Clip2 Reflector, a supernode for the Gnutella network. The Reflector can also explicitly act as a proxy for a slower peer by caching its list of available resources and responding to requests for those resources.

The continued interest and wide deployment of the Gnutella network is evidence of its effectiveness, and its radical departure from the traditional client-server network

architecture has prompted much academic study. A 2000 paper by Xerox Parc researchers appropriately applied the term ‘freeriding’ to the behaviour of most hosts in the Gnutella network at the time: the researchers found that 70% of the users shared no files, 90% answered no queries, and 50% of all queries were returned by 1% of the hosts [AH00]. The developers of the many peers must have noticed this behaviour as well, as it soon became common among the different implementations to refuse to honour queries from web-based clients that by their nature were not contributing content back to the network.

A mathematical analysis of node degrees and hop counts [Rit01] concluded that the Gnutella network would not scale, however this analysis assumed a constant number of connections per host across the network and a uniformly random construction. In practice, the number of connections a Gnutella peer maintains varies widely, and the networks themselves have been shown to exhibit strong small-world characteristics as opposed to being uniformly random [Jov00].

The Gnutella protocol itself makes no explicit attempts to generate a small-world network, so the reason for the emergence of these characteristics remains unclear. It has been speculated that a Gnutella network may grow to reflect the small-world social networks of the users administering the peers [IRF02], although in practice a Gnutella user has very little control over the construction of the network. More plausibly, Gnutella’s broadcast advertisement and response mechanism would necessarily reveal closer neighbours before further neighbours, and the relatively low node degrees in the network would cause more local connections and an increased clustering coefficient rather than longer-range connections.

Where small-world networks exhibit ‘cliquish’ topologies, others have looked at

the distribution of node degrees to determine whether the node degrees follow a ‘power law’ distribution⁴. Ripeanu has argued [Rip01] [RFI02] that Gnutella networks are not pure power law networks, but that they show much of the robustness that characterizes power-law networks without so many of the weaknesses. (A primary weakness of power law networks is that there are a small number of high degree nodes which provide much of the connectivity in the network and are vulnerable as central points of failure.) This result is disputed by Gummadi et al. [GSG02] who argue that an orchestrated attack against the 4% of nodes with the highest degrees is enough to shatter the overlay into many disjoint pieces. It is worth noting, however, that 4% of the peers in a typical Gnutella network is a significant number, and that successfully attacking this number of independent machines is not a trivial undertaking.

Adamic et al. [AHL01] show that search in a power-law network can scale sub-linearly with the number of hosts in the network by adopting some simple rules. In their model, each peer maintains a list of the resources available from all of its neighbouring peers within a two-hop horizon, and a flooding search then need only query the peers with the highest node degree. They cite a study by Clip2.com [Cli00] showing a rough power-law distribution of node degrees in a Gnutella network, and conclude that this is a viable method for improving the scalability of searching in a Gnutella network.

As in the work above, Lv et al. [LRS02] propose an adaptive load-balancing scheme that adjusts the topology of the overlay network in response to traffic flows and favours

⁴A power law distribution is characterized by many small occurrences and few large occurrences, and has been observed in many human-made and naturally-occurring systems such as city populations, incomes, and earthquake magnitudes [Ada].

high-degree peers when searching. The authors' goal is to utilize the significant heterogeneity present in Gnutella networks to achieve a scalable search strategy.

Another approach to improving the scalability of a flooding broadcast search is to reduce the branching factor of the broadcast. At the extreme, a peer receiving a search request could forward it to one other randomly-chosen neighbour and achieve a scalable but slow random-walk search. Lv et al. [LCC⁺01] propose a branching factor of k at each receiving peer to trade off between speed and scalability and find that their model scales much more favourably than naïve flooding, but provide no guidance on choosing k . Their model also requires that peers periodically check with the originator of the search to determine whether to continue, but they do not address the overhead, NAT⁵ and firewall issues inherent in such out-of-band communications. This reduced branching factor approach has been taken up elsewhere, and a paper by Kermarrec et al. [KMG00] analyzes the branching factor in terms of the resulting number of peers that will not be reached by the message. They conclude that the branching factor for a network with n peers should be $\log n + c$ where c is a constant 'safety' factor. Their analysis shows that the probability of failure decreases very quickly to almost zero with $c > 6$.

In a response to claims that Gnutella would not be able to scale, another analysis [Gun02] showed that with an intelligently-built network in the form of a hypercube or a hypertorus, the search horizon could be manageable and the bandwidth required could scale nearly linearly with the number of hosts. This is an intriguing line of

⁵Network Address Translation (also known as IP-masquerading) is a common technique used to enable machines on a subnetwork to access the Internet at large through the single IP address of their gateway to the Internet. A peer residing on the gateway can provide a communications bridge between the subnetwork and the Internet, but out-of-band communication between the subnetwork and the Internet is usually not possible.

thought, but maintaining such an arrangement in a decentralized manner in the face of a high peer turnover rate would be extremely difficult.

In another study, Markatos found that many queries in the Gnutella network were repeated and showed that caching of query results could result in up to a 200% improvement in speed. This was confirmed independently by Sripan [Sri01].

Gnutella networks have also been criticized [GSG02] for not giving any consideration to the physical topology of the Internet, leading to ineffective use of the infrastructure.

The Gnutella approach to a pure peer-to-peer flooding network has been adopted elsewhere. In particular, Texar's **s-peer** builds a network that is functionally identical to Gnutella's in its resource discovery and routing behaviour. One difference is that s-peer attempts to improve connectivity by 'short circuiting' the network with connections to nodes that are topologically far away.

Sun's **JXTA** project is an ambitious attempt to develop a flexible substrate for any peer-to-peer network. A review by O'Reilly states that, "JXTA's initial design represents an educated guess at the makeup of a robust P2P foundation." [Tru01]. JXTA is a flooding broadcast peer-to-peer network at its core, but it attempts to allow maximum flexibility in the services implemented on this base. The JXTA search reference implementation involves supernodes ('search hubs' in JXTA parlance), but resource discovery can be achieved through flooding broadcast on a local ethernet, individual invitation, cascading via other peers, or via centralized rendezvous points [Sunb].

Routing in JXTA is equally flexible. Messages can carry their own routing information in the form of a complete list of nodes to traverse, or they can just carry the

name of their destination node and leave the routing decisions to the nodes themselves. In the latter case the nodes would query their local cache or a peer router to find a route to the destination [Sunc].

Another example of a pure flooding broadcast network is **Adaptinet** [Ada01]. Adaptinet is a fully-decentralized peering infrastructure that follows some simple heuristics to adaptively modify its network topology: peers periodically probe the network to measure response times to other known nodes, and acquire faster connections and shed slower ones.

The **FastTrack** network uses proprietary peering software written by the KaZaA BV company and licensed by Grokster and KaZaA. the FastTrack network is technically notable as a large and efficient flooding broadcast network that makes heavy use of supernodes. The peering software examines the resources available to the host it is running on and automatically elects itself to be a supernode if its resources allow. Search queries are broadcast only to other supernodes in the network.

The FastTrack network is also interesting as a test case in the legality of decentralized file-sharing networks. The primary benefit of a decentralized network in a file-sharing context is its resistance to legal prosecution. The Recording Industry Association of America (RIAA) had not initially attempted to prosecute FastTrack or any of its licensees, and seemed to view the network as similar to the Gnutella network in which there was no central administration to prosecute. However, two sequences of events drew attention to a key difference between FastTrack and Gnutella. The first came about after the FastTrack protocol was reverse-engineered and implemented in the giFT peer (Gnu Internet File Transfer, or more playfully in the Gnu tradition of recursive acronyms, giFT Isn't Fast Track). The presence of an unlicensed (and

non-paying) participant in the FastTrack network was obviously not to KaZaA BV's liking, and they moved to require the use of a centrally-delivered encryption key to log on to the network. The second event came about because of a dispute between Morpheus, an initial licensee, and KaZaA BV. KaZaA BV blocked Morpheus's participation in the network through the use of the same centrally-delivered encryption key mechanism. These events led the RIAA to conclude that the FastTrack network was indeed under central control, and they promptly brought suit against KaZaA BV. Interestingly, a Dutch court has recently ruled [Bor02] that KaZaA BV is not liable for the content that users of the network exchange. Whatever the eventual outcome of these proceedings, it is clear that a peer-to-peer network's ability to devolve legal responsibility to all involved is a key factor in their widespread popularity.

The **Anthill** project [Mon01] [BMM01] is an interesting approach to developing a resource discovery strategy in a decentralized network. The network is built on the JXTA substrate and is a swarm system with an ant colony metaphor. An 'ant' is dispatched for each resource location request, which means that the ant's code is transferred to a neighbouring 'nest' (another peer) and run. The ant (i.e. the mobile code) is able to move to another nest, query the new nest's local storage, add to the new nest's storage to inform it about other nests, obtain the list of neighbouring nests, get and set pheromone information, insert new resources into the new nest's local storage, and notify the local nest that the ant has returned and completed its search.

Flooding broadcast behaviour is only one of the possible strategies that the Anthill network allows, and the authors have developed a simulated file-sharing peer called

Gnutant that employs a genetic algorithm to tune the parameters (such as the searching versus following probability) of the ant agents. Gnutant uses the proximity of hashed keywords to achieve routing between neighbours, and the authors show that Gnutant's routing compares favourably with Freenet's. (Hashed keywords are a form of distributed hash table, described fully in section 2.2.4).

Grid Computing is an attempt to make distributed computing resources available for use with the same ease that power from the electrical grid is consumed. Grid computing focuses on two areas: resource discovery, and accounting methods for preventing abuse and gaining compensation for contributed resources.

In one approach [IF01], nodes selectively forward requests for resources using one of four methods: random forwarding, past experience of forwarding similar queries, 'best' neighbour, and past experience combined with the best neighbour. The authors show that the flooding broadcast techniques given are an effective way to achieve resource discovery in a decentralized network.

The **Free Haven** project [DFM00] was developed with anonymous persistent storage as its primary objective. The network consists of a decentralized collection of Free Haven servers which collectively store all the data placed in the network, though the data is encrypted and distributed so no server individually knows what it is storing or who owns the data. Participation in the network is motivated by the desire to store data that is persistent in the face of a foe desiring to destroy it. In this context, encrypted local storage is not sufficient.

The resource location aspect of this network is mundane: a 'fingerprint' of the desired file is communicated to the network as a whole using a flooding broadcast, and matching segments of files are transferred to the requester until the original file

can be reconstructed. This example serves to show the wide variety of needs that can be met with flooding broadcast networks.

There are many other flooding broadcast networks. **Konspire** is a file-sharing application that hosts a replicated index on a set of supernodes. **G Diffuse Networks** is an experimental substrate written in Python to explore peer-to-peer protocols. **Gnougat** is similar to Gnutella in functionality except that it uses hashes instead of resource names to reduce the problem of corrupt files or intentionally mis-labeled advertising. **Mojo Nation**'s unique contribution is to use a trusted third-party to trade a currency called 'mojo' between peers to encourage participation. A packet network algorithm proposed by Karp and Kung [KK00] routes packets to neighbouring nodes that are geographically closer to the packet's destination. In this protocol, if no 'closer' node can be found, the node routes 'to the right' to begin a circumnavigation of the destination node with the expectation that another node will be able to find a successful route.

In conclusion, we have seen that flooding broadcast networks have been successfully applied in a wide range of contexts. Though they have significant drawbacks in terms of efficiency, they are extremely resistant to technical failures and legal prosecution.

2.2.4 Distributed Hashing Systems

Distributed hashing systems are in some sense the inverse of flooding broadcast networks. In a flooding broadcast network, resources exist on their hosts and requests propagate outwards in an attempt to query as many hosts as possible. In a distributed hashing system, each resource resides in a canonical location in the network and requests propagate in a straight line towards the resource's home.

The obvious benefit over flooding broadcast networks is the tremendous reduction in the amount of traffic required to satisfy any query: any item can be located (or its non-existence established) in a bounded number of hops. This number of hops typically grows as $O(\lg n)$ with the number of nodes in the network, so the network is thus highly scalable.

A document's location in the distributed hash table is based on a canonical identifier, which is typically a SHA-1 hash of the document. This leads to perhaps the biggest disadvantage of a distributed hashing system when compared to a flooding broadcast network: a query can only be in terms of the resource's identifier and not on keywords, fuzzy matching, the full text, or any other attributes of the resource. This compares unfavourably with flooding broadcast networks' ability to query on any aspect of a resource, to the extreme of allowing for infinitely customized queries expressed in mobile code. A further result of this arrangement is that a distributed hashing system cannot be browsed in the same way that a flooding broadcast network can. In a flooding broadcast network users quickly learn that if a peer provides a successful query for one resource, it is often worthwhile to browse the other resources that peer is offering. This is impossible in a distributed hashing system.

Another relative weakness of a distributed hashing system is its increased vulnerability to malicious activity. A rogue peer in the network can misdirect queries, insert bogus data and flood the network with queries just as it can in a flooding broadcast network, but since the success or failure of a request is predicated on that single request successfully negotiating the path to the resource's location, any malicious peer along the path can effectively sabotage the query. In this way, attacks in a distributed hashing system are amplified beyond the bandwidth and computing cycles needed to

perform them.

In a flooding broadcast network, no work is required to add new resources to the network: a peer simply services requests as they arrive. In a distributed hashing system, each resource that a peer is offering must be announced to the network so that the resource (or a reference to it) can migrate to the resource's canonical location in the network. For this reason, distributed hashing systems may be inappropriate in contexts where the number of resources in the network greatly exceeds the requests for those resources.

Distributed hashing systems have a wide range of potential applications. They could be used for cooperative mirroring of information in a decentralized manner, time-shared storage of information, distributed indices as would be required for a file sharing application, and combinatorial searches such as mapping sections of a key-space to the machines responsible for processing them.

The **Chord** project [DBK⁺01] [SMK⁺01] is an elegant effort to realize a distributed hashing system. The Chord system itself is a substrate whose only function is to map 'keys' to peers in the network. The keys, as described above, are canonically derived from the resources they represent and the mapping is done using 'consistent hashing' [KLL⁺97]. This mapping projects the nodes into a one-dimensional closed space, so each node can be thought of as occupying a position on a circle. Each node thus has two neighbours in this space, and the only routing decision is in which direction around the circle the message should be passed. Consistent hashing has the desirable property that the addition or removal of peers in the network disrupts only the mappings local to the changed peer, varying as $O(1/n)$ in an n -node network. Thus the typical host churn of a peer-to-peer network results in relatively little

movement of keys.

Chord has three features that distinguish it from other distributed hashing systems: simplicity, provable correctness and provable performance. The consistent hashing function requires routing information from only a few other nodes to resolve the hashing function and make a routing decision. In an idealized steady-state n -node system, nodes need knowledge of only $O(\lg n)$ other nodes and can resolve lookups in $O(\lg n)$ messages to other nodes. A node joining and leaving results in $O(\lg^2 n)$ messages with high probability. There are many benefits to Chord networks: the hashing function effectively achieves load-balancing of files (and therefore file-accesses) across the network, decentralization, scalability, availability of data and flexible naming.

There are some weaknesses as well. If multiple peers leave the network quickly the network can become partitioned and Chord does not explicitly address this possibility beyond the authors' suggestion that each peer could maintain a small list of random nodes and periodically ping for them. Chord networks are also susceptible to malicious or buggy peers, although the authors stress that this is a threat to data availability and not data authenticity because authenticity is protected using public key cryptography. The network allows nodes to insert themselves at any point, so to sabotage a specific data item a node could insert itself as the successor to that data item and then fail to store the item when asked to.

Chord is similar to Freenet (described below), but Freenet's focus is on anonymity and it provides no guarantees on success or performance. Chord's routing is also similar to Plaxton's algorithm (used in Tapestry, described below), but Chord is substantially less complicated and handles concurrent node joins and failures well. In a file-sharing context, Chord could address the lack of scalability in flooding broadcast

networks such as Gnutella because of those networks' use of broadcasts.

An analysis of Chord's suitability as a replacement for DNS [CMM02] illustrates many of the tradeoffs inherent in distributed hashing systems. In a Chord system where public-key cryptography is used by domain administrators to certify subdomains and DNS information is served by the network as a whole, Chord has the advantage of providing definitive answers in a bounded amount of time (one study found that 23% of DNS lookups went unanswered and 13% of lookups resulted in a negative response, due in part to loops in the servers' resolution mechanism and to records that point to incorrect hosts [JSBM01]); Chord has a more robust configuration mechanism (DNS servers don't verify that their parent's server refers to them, where Chord would report an error while verifying the signing of the parent's key); Chord would be far more robust to host failures and denial of service attacks; and Chord would eliminate the significant propagation delays inherent in DNS.

However, Chord has some significant drawbacks as a replacement for DNS. In the case where a subnetwork became disconnected from the rest of the network (possibly through the loss of its physical connection), DNS would allow the subnetwork to continue to resolve names within the subnetwork while Chord would not. Chord would also result in much higher latencies than DNS's locally-served results. From a social perspective, the use of Chord would require the publishers of DNS names to rely on other servers in the network to serve those names, with no incentive to actually run a server of one's own. Finally, and most compellingly, Chord would not allow for any dynamically generated responses. The current DNS implementation allows for much flexibility in serving responses: wildcards can be used to map a range of names to a single address to avoid publishing internal host names (`*.carleton.ca`, for example),

or randomly rotated addresses can be returned to achieve load-balancing [Bri95]. In the case of Akamai, strategically-generated addresses are returned to achieve load-balancing or to route clients to nearby servers [Mah01].

This example shows that while Chord and decentralized hashing systems in general are advantageous in certain contexts, in the context of a replacement for DNS they are found lacking. A primary advantage of decentralized hashing systems is their freedom from central administration, but in the case of Chord/DNS, a global administrative body would still be required to cryptographically certify top-level domain names. Another advantage of hashing systems is their low latency compared to flooding broadcast peer-to-peer networks, but DNS's distributed index is far more efficient. And finally, where DNS allows for tremendous flexibility through dynamically-generated responses, distributed hashing systems are restricted to single-key lookup with no mechanism to achieve similar effects.

The **Cooperative File System** (CFS) [Dab01] is a read-only file system implemented on top of a Chord network. 'Read-only' in this context means that files can only be modified by their owner, and can only be replaced as a whole. CFS works by breaking files into blocks and distributing those blocks to their appropriate hosts in the network. This means that CFS is inherently load-balanced because the hashing function distributes the blocks uniformly across the network. It also contrasts with Freenet and PAST (described below) which store whole files and therefore have the potential problem of individual servers running out of space when assigned large files when the network as a whole has sufficient free space to store the files. This load balancing inherently provides high availability for popular data items, which contrasts with file-sharing networks such as Gnutella or Napster which do not explicitly

replicate but instead rely on users to copy and republish popular items.

As with Chord, CFS boasts provable efficiency, provably fast recovery times after failures and scalability. Also as with Chord, CFS does not address abuse of the network, anonymity, or any form of network locality to improve routing.

The **Freenet** project [CSWH00] is a distributed hashing system whose primary goal is anonymity. A Freenet node holds three things: keys (which are global resource locators, analogous to URLs), addresses of other nodes that are likely to know about similar keys and (optionally) the data corresponding to the keys. A node propagates a search request by forwarding it to another node that it thinks is most likely to have the key. In this way, requests for keys are passed from node to node through a chain of proxy requests where each node makes a local decision about where to route the request next. Successes and failures are both reported back along the path the request took. When a failure is reported, the upstream node gives it to the next best downstream node, so the search is a steepest-ascent hill-climbing search with backtracking.

A node is listed in other nodes' routing tables with a specific key, so the node will tend over time to receive requests for similar keys. Data obtained through backtracking is also cached, so nodes will amass content for similar keys. Because of this, the authors hypothesize that the quality of routing will improve over time as the nodes increasingly specialize in locating certain types of keys.

Freenet's focus is on anonymity. It also provides deniability (the ability for hosts to deny knowledge of the content they are hosting), resistance to denial-of-service attacks, efficient dynamic storage and routing, and decentralization of all network functions. It is worth noting that Freenet's anonymity specifically applies to file

transactions, and not to network usage in general.

Freenet suffers from the common problem among distributed hashing systems that searches can only be in terms of the canonical identifier for the resource. The Espira and Frost projects aim to address this problem in Freenet by maintaining catalogs that allow for full searching of metadata, but maintenance of these catalogs is problematic as they either require a single globally-known location or some form of global propagation mechanism.

Another interesting and highly flexible distributed hashing system similar to Chord is the **Content Addressable Network** (CAN) project [RFH⁺00]. In CAN the hash table is mapped to a d -dimensional space and each node adopts responsibility for a portion (a ‘zone’) of this space. When a new node joins the network, it randomly decides which zone it belongs to and navigates its way to that zone. The random choice ensures uniform coverage of nodes in the hash table space. The current occupant of the zone divides the zone in half, and both nodes inform their neighbours of the changes. Requests for a resource are also mapped to a zone by the hashing function, and the request propagates in a straight line through the d -dimensional space towards the zone that the resource would reside in.

When a node leaves the network, its zone is either merged or paired with a neighbour’s zone, and that neighbour handles either the new merged zone or the pair of zones. Neighbours periodically reaffirm their presence to each other, and when a neighbour hasn’t responded for a threshold amount of time it is presumed dead and the neighbouring nodes collaborate to decide who will assume the dead node’s zone.

For a d -dimensional space partitioned into n equal zones, the average routing path length is $\frac{1}{4} \cdot dn^{1/d}$ hops and individual nodes maintain $2d$ neighbors. This means that

for a d -dimensional space, the number of nodes (and therefore zones) can grow without increasing the per-node state while the average path length can grow as $O(n^{1/d})$.

There are many alternative routes for a request to take towards its destined zone, so nodes can route messages around other nodes that have ceased functioning. If all of the neighbours in the desired direction disappear, the routing falls back on stateless controlled flooding until a node that is closer to the destination is found.

It is interesting to compare a CAN to other resource-discovery techniques. The distance vector [Hed88] and link state [MRR80] algorithms used in IP routing require widespread dissemination of local topology. These are well-suited to IP routing where topology changes are infrequent, but not for the typical overlay peer-to-peer network in which node churn is high. Plaxton's algorithm [PRR97], another distributed hashing technique mentioned earlier, is used in OceanStore (discussed later) and web-caching arrays with fairly stable hosts. It only scales on the order of thousands of nodes however and requires global knowledge, so CAN compares favourably with it. And unlike FreeNet or Gnutella, content in a CAN is always available: Freenet nodes may have an inconsistent view of the network, and data may be beyond the horizon of a Gnutella node's search.

A CAN is an extremely flexible arrangement, and its originators have proposed numerous modifications and tradeoffs that can be made. A simple modification is to increase the dimensionality of the space the hash function is mapped to. This would increase the number of neighbours that each node maintained, and decrease the routing path lengths. The authors propose multiple 'realities', which are alternative hashing functions giving rise to alternative d -dimensional spaces. In this scenario, a node would exist at points in multiple spaces at the same time, so a routing decision

would involve determining which ‘reality’ the destination node was closest in and propagating the message in that reality. This would add redundancy because the hash tables would be replicated in each reality, but if increased neighbours and increased local storage were not a constraint then this would be an appropriate modification. It could also provide increased redundancy in routing: if routing failed in one reality, it could be moved to another. The effects of these modifications would be that increasing dimensionality would improve routing efficiency at the expense of a higher per-node overhead to keep track of more neighbours, and multiple realities would improve data availability and fault-tolerance.

The latency of neighbours could also be measured, and instead of forwarding a request to the neighbour with the shortest Cartesian distance in the d -dimensional space, the request could be forwarded to the neighbour with the maximum ratio of distance to latency. This would introduce some consideration for the underlying topology of the network, which would reduce total routing times. Another method for introducing some sensitivity to the Internet topology would be to identify m globally-known landmarks and have each node rank the landmarks according to the measured latency from that node. If the hash space was then partitioned into $m!$ zones (for the $m!$ orderings of the landmarks), then neighbouring zones would have close to the same landmark-latency times and would presumably be topologically close on the underlying network. This is similar to another proposal [RHKS02] to introduce a coarse-grained binning strategy associating topologically-close nodes to improve the efficiency of communications in a peer-to-peer network.

Another modification would be to allow more than one node to occupy a zone, so a zone would be split only when the number of nodes in it reached a threshold.

A node's list of neighbours would include a single, randomly chosen node from each neighbouring zone as well as the list of all nodes in the same zone. Nodes would periodically ask their neighbouring nodes for the list of nodes in the neighbour's zone and test the latency to each of those nodes. The node in the neighbouring zone with the lowest latency would be retained as the new neighbour. In this arrangement the hash tables could be replicated across all nodes in the zone or split between them, the tradeoff being redundancy and fault tolerance versus size of the stored data. This zone-overloading approach would reduce overall path length in the network because there would be fewer zones. It would also reduce per-hop latency because the nodes would be searching for low-latency neighbour, and it would improve fault tolerance because all nodes in a zone would have to crash simultaneously to affect the data availability. However, the approach would significantly increase the complexity of the system and the amount of control traffic required.

A final suggested modification is for nodes in a CAN to cache requests and serve responses from their cache, and additionally to 'push' copies of a popular key out to their neighbours to cache.

A number of similar distributed hashing systems also exist. **Tapestry** [ZKJ00], on which OceanStore and Silverback are built, is significant for its use of a modified version of Plaxton's algorithm [PRR97] for routing. Plaxton's algorithm assigns each node a fixed-length label and performs routing by hashing requests into the same space as the node labels and forwarding the requests to neighbouring nodes that match the request's label in one more digit than the current node. In this way a request is increasingly 'resolved', and reaches its destination in a bounded number of hops. It requires a routing table that is $O(\log n)$ in size, and as with other hashing

functions it routes in $O(\log n)$ hops. Plaxton’s algorithm was originally proposed for a web-caching environment and is criticized [RFH⁺00] in the context of a widely distributed peer-to-peer network for not scaling to millions of nodes, for requiring global knowledge of all nodes to achieve effective routing, for not providing a mechanism for nodes to discover each other in a decentralized manner, and for not dealing well with significant node churn because each arrival or departure affects a logarithmic number of other nodes. Tapestry’s modifications to the algorithm address these concerns and allow for highly dynamic node populations. OceanStore itself is an infrastructure for persistently storing data on a large-scale decentralized network of untrusted servers.

PAST is built on **PASTRY** [RD01], a distributed hashing system substrate. SCRIBE is also built on PASTRY, and builds a multicast tree in the network to achieve a decentralized publisher-subscriber mechanism. Tapestry and PASTRY both exploit locality by measuring the proximity between pairs of nodes and by choosing nearby nodes for their neighbours

The **Ohaha** system used a hashing-like algorithm and Freenet-style routing, but it has since become defunct.

Distributed hashing systems have been criticized as a step backwards in terms of searching, but some recent work [HHH⁺02] has attempted to bring flexible searches to distributed hash tables. The general approach in this work is to break all resource identifiers into bigrams or trigrams and index these sub-identifiers separately in the network. For example, ‘Beethoven’ would be indexed under each of the trigrams ‘Bee’, ‘eet’, ‘eth’, ‘tho’, ‘hov’, ‘ove’ and ‘ven’. A query in this hashing system would be broken into similar trigrams, and potential query hits would be ranked by the number of sub-identifiers they match. This would obviously increase the network

traffic required to satisfy a given query, but the authors claim that involved database techniques such as pipelined hash joins can be used to reduce this traffic. Further work is necessary in this area to bring the query facilities of distributed hashing systems up to the minimal requirements of partial-text or other fuzzy searches.

The Internet has achieved widely-distributed point-to-point routing with remarkable efficiency, but despite the well-defined need for efficient multicast routing there continues to be no widely adopted low-level mechanism to achieve this. A case has been made for multicast at the overlay network level [CMB00] [CRZ00] and indeed successful solutions such as Akamai have shown that application-level content-delivery overlays are a viable approach. Interestingly, all major distributed hashing systems have proposed multicast as an appropriate use of their substrates. A prototype multicast system built on Chord [SAR⁺02] reveals that it can achieve routing latencies within a factor of two of the optimal routing on the underlying network. A multicast application for a CAN [RHKS01] exists, as does Bayeux [ZZJ⁺01], an application-level multicast scheme built on Tapestry. These are interesting not only as illustrations of the usefulness of distributed hashing systems, but also because of their incorporation of flooding broadcast behaviour into distributed hashing systems: in all cases, multicast is achieved by a controlled flooding broadcast outwards from the source peer, relying on the pre-existing structure of the distributed hashing system instead of requiring a computationally- and bandwidth-expensive overlay tree to be constructed.

In conclusion, distributed hashing systems offer an elegant and efficient alternative to the traffic required by flooding broadcast networks, at the expense of increased complexity and inflexible searches.

2.2.5 Alternatives

Most resource discovery strategies in decentralized networks fall into the categories above, but there are a number of interesting alternatives that are worth mentioning.

The **Alpine** (Adaptive Large-scale Peer2peer Information NEtworking) network [Pec02] is an interesting approach in which every peer maintains a lightweight persistent connection to every other peer in the network, so every peer is an immediate neighbour of every other peer and broadcast to the entire network is achieved in one hop. This is done with UDP messaging on one port, and the author claims that 50 000 connections can be maintained with only 12 megabytes of memory.

As already stated, none of the approaches discussed thus far have adequately dealt with abuse of the network by malicious peers. Alpine's author feels that this problem cannot be addressed by purely technical means, and so emphasizes a 'social discovery' approach that is analogous to social interactions among people. Social discovery implies direct, continued interaction between peers, and to achieve this Alpine connections span disconnections, IP address changes and NAT routing changes. This allows an Alpine peer to evaluate the past history of another peer, do reputation management, and identify malicious peers or peers who aren't contributing to the network. Searching in the network is optimized by ranking peers based on past history. This arrangement encourages peers to contribute, because they will otherwise be pushed out of the network.

Globe [HvST97] and **TerraDir** [SBK01] are distributed systems that exist outside of flooding broadcast or distributed hash networks. In this scheme, resources are organized into a hierarchical tree structure and each peer hosts one vertex in the tree. Lookups are done by walking up the tree and descending a different branch to

the resource's host. An advantage of this approach is that query hop counts vary linearly with the height of the tree, which is $O(\log n)$ with the number of peers in the network. Because each host volunteers itself for a specific part of the network, locality in the network can be exploited. This is in contrast to distributed hashing systems, where the uniformity of the hashing precludes the use of network locality to guide organization. The approach has some obvious downsides, however: a hierarchical directory of information requires hand-building and is not appropriate for many types of data; as well, with only one route to each resource, fault tolerance becomes a critical factor. Globe does not address fault tolerance, and TerraDir addresses it through a recovery protocol and constant caching of a peer's contents elsewhere in the network. Globe positions itself as a lookup service for mobile hardware; such tree-based systems are not appropriate for general resource discovery problems.

The **Groove** network is a 'groupware' application that allows people to collaborate and communicate within a virtual shared space. Groove takes an opportunistic approach to communication, and can operate in a centralized or decentralized fashion depending on its configuration and environment. For example, on a local ethernet where efficient broadcast could be achieved, Groove would operate in a flooding broadcast peer-to-peer style. In a larger network such as the Internet, Groove could take advantage of centralized servers to facilitate communication.

Sun's **Jini** project [Suna] creates a network infrastructure in which clients (usually mobile devices) can find services (such as a provider to host them). The network uses a combination of multicast announcement, request, and unicast response protocols for both clients and service providers to find supernodes ('lookup services') on the network. If no lookup service can be found then clients can do a 'peer lookup' by

	Centralized Indices	Decentralized Indices	Flooding Broadcast Networks	Distributed Hashing Systems
scalability	poor	good	poor	good
flexible searching	yes	yes	yes	no
single point of failure	yes	yes	no	no
distributed cost	no	somewhat	yes	yes
peers queried	1	1	$O(n)$	$O(\log n)$
peers affected by addition or removal of resource	1	n_{hosts}	1	$O(\log n)$
robustness to host failure	poor	average	excellent	good
bounded search completion	yes	yes	no	yes

Table 2.1: A comparison and ranking of the approaches presented in this chapter. The variable n refers to the number of nodes in the network, and n_{hosts} refers to the number of nodes hosting the distributed index.

broadcasting a request for a service and choosing from the responses.

2.2.6 Resource Discovery Summary

In this chapter we have seen a wide variety of strategies for achieving resource discovery in decentralized networks. Centralized or decentralized indices are an obvious solution but pose scalability and single-point-of-failure problems. Distributed hashing systems use highly structured networks to achieve very efficient searching, but do not deal well with highly transient peers and do not allow for flexible searching. Flooding broadcast networks provide infinitely flexible searches and allow for transient nodes, but are limited by a simplistic and inefficient propagation mechanism. These factors are summarized in Figure 2.1.

The two most promising approaches to organising a peer-to-peer network then are

flooding broadcast networks and distributed hashing systems. Distributed hashing systems, despite their great promise, are complicated and untested in the real world and currently exist only within the realm of academic research. On the other hand, flooding broadcast networks (including Gnutella and FastTrack/KaZaA) comprise the vast majority of deployed peer-to-peer networks, and despite their scaling difficulties are approaching the participation levels of the original Napster. We have chosen, therefore, to build on this base in an attempt to improve on current resource discovery strategies in flooding broadcast networks.

2.3 Genetic Programming

Genetic programming [Koz92] is a stochastic evolutionary process for automatically generating expressions. The expressions represent solutions in a given problem domain, and genetic programming uses this evolutionary process to recombine existing solutions into better solutions.

We will adopt the use of genetic programming in later chapters to guide our search through the protocol space described by the flooding broadcast meta-protocol (developed in Chapter 3) in pursuit of optimal flooding broadcast network protocols.

2.3.1 The Process

The expressions used in genetic programming consist of terminals and operators that have meaning in the problem domain, and are typically expressed as LISP s-expressions or as n -ary expression trees. For example, a mathematical expression representing the equation $2x^2 - y$ could be represented as `(- (* 2 (pow x 2)) y)`, or the logical hypothesis $A \rightarrow \neg B$ could be represented as `(implies A (not B))`.

The evolutionary process begins with the generation of an initial population of expressions. This initial population is generated stochastically and thereafter the expressions are manipulated by evolutionary operators to produce new expressions, so it is important that the operators and terminals themselves be robust in terms of their inputs and that all possible arrangements be considered syntactically correct.

Each expression in the initial population is evaluated according to a domain-specific fitness function. This fitness function assigns each expression a fitness ranking, allowing the expressions to be ranked within the population. In the case of a regression analysis application for example, the fitness ranking might represent the cumulative deviation of the expression from the desired regression data. In this case, a ‘fitter’ expressions would more closely approximate the given regression data.

Expressions are then chosen according to a selection scheme, and recombined (‘bred’) to form the next generation of expressions. The selection scheme can involve any of the following methods: *roulette selection*, in which expressions are selected with a probability proportional to their fitness ranking; *tournament selection*, in which tournaments of (typically) two to five expressions are randomly selected and the fittest expressions within each of the tournaments are selected; and *elitism*, in which the fittest expressions are allowed to pass unchanged into the next generation. Other selection methods exist, and the biological metaphor is stretched even thinner in arrangements such as *steady state* populations in which only a portion of the population is replaced during each generation.

The recombination of the chosen expressions is done with an analog of the crossover and mutation operations that recombine DNA during biological reproduction. These operations are performed with a fixed probability.

Figure 2.2: The Crossover Operation. The subtrees $t1$ and $t2$ are randomly selected from each of parents A and B . These subtrees are then exchanged to produce two new offspring C and D .

Figure 2.3: The Mutation Operation. The subtree $t1$ is randomly selected from the single parent A and replaced with another randomly-generated subtree $t2$ to produce the descendant B .

With the crossover operator, one node in each of the parents' expression trees is picked randomly and the subtrees rooted at each of these nodes are exchanged. This forms two new children with genetic material related to, but different from, their parents. The crossover operator is illustrated in Figure 2.2. Crossover recombines portions of other successful expressions, so the evolutionary search through the problem space is guided by previous successes.

With mutation, one node in a single parent's expression tree is picked randomly and the subtree rooted at this node is removed and replaced with a new, randomly-generated tree. This process is illustrated in Figure 2.3. Mutation is meant to introduce diversity into the population, although the probability of an operator or terminal disappearing entirely from the population is negligible. For this reason, mutation plays a far less important role in genetic programming than in the related field of genetic algorithms [Koz92].

The new expressions (the 'offspring') resulting from these evolutionary operations form the next generation of expressions, and the entire process is repeated. This continues until some criterion for stopping is met. This may be: the discovery of a perfect solution; a loss of diversity in the population of expressions; the rate of improvement of the average or best fitness falling below a given threshold; or simply the completion of a fixed number of generations.

2.3.2 Genetic Algorithms

Genetic programming is closely related to the field of genetic algorithms [Hol75]. Genetic algorithms in fact operate in a manner identical to that described in Section 2.3.1, except that genetic algorithms typically encode their solutions in a fixed-length binary genome in place of the variable-length, tree-encoding genome used in

genetic programming. Consequently genetic algorithms also require crossover and mutation operators that operate on these fixed-length strings in place of the tree operators used in genetic programming.

2.3.3 Benefits

Genetic programming has the benefit of producing human-readable expressions that can be analyzed and used independently of the genetic programming environment. This contrasts with techniques such as neural networks, which are often used to solve similar problems but which produce solutions that cannot easily be reverse engineered to determine why they work or to provide insight into the problem domain.

As well, because genetic programming is a stochastic process that begins with a completely random initial population, it is less subject to any biases that might be introduced by a human's search through the problem space for solutions. As mentioned in Section 1.3 no process is completely free of human bias, however the solutions arrived at through genetic programming can be considered optimal within the problem space and with respect to the given fitness function.

2.3.4 Successes

Genetic programming has been shown to be effective in a wide range of problem domains, including robot guidance [Iba97] [FN97], satellite guidance [How96], data mining [RE96], neural net design [GWP96], circuit design [Tho96] [KABK96] and economic strategy development [Ash97] [Len97].

It has produced expressions that are on par with the best solutions created using other methods (including hand-coding) on problems such as creating cellular automata that solve classification problems [ABK96].

Koza describes a study in which genetic programming was used to create new designs for lowpass filters⁶ [BKKA99]. The genetic programming system exhibited human-level creativity and design skills by arriving at lowpass filters which corresponded to four inventions that were patented in the period 1917–1936. Koza elsewhere presents 25 cases in which genetic programming has produced results on a par with human designers [Koz99].

2.4 Conclusion

This chapter began with an introduction to the concept of small-world networks, which later in the chapter informed the discussion of the Gnutella protocol and will continue to aid in evaluating new network protocols in the remainder of this thesis.

Small-world networks were followed by a discussion of a number of different approaches to the resource-discovery problem in peer-to-peer networks. This discussion ranged over centralized and distributed indices, flooding broadcast networks, distributed hashing systems and some alternative and hybrid methods. This section reached the conclusion that distributed hashing systems, while promising, are complicated and relatively untested. Flooding broadcast networks, on the other hand, are widely deployed and effective but suffer in general from inefficient resource-discovery techniques. The chapter concluded with an introduction to genetic programming.

The following chapter brings flooding broadcast networks and genetic programming together by developing a meta-protocol to describe a broad class of flooding broadcast network protocols, and then proposing the use of genetic programming

⁶Lowpass filters are analog circuits that filter out all but a specific range of bandwidths.

to guide a search for optimal protocols through the space described by this meta-protocol.

Chapter 3

Approach

Our main goal in this thesis is to develop a flooding broadcast meta-protocol (FBMP) capable of describing a wide range of possible flooding broadcast network protocols, and to show that genetic programming is a viable method for deriving specific network protocols from this meta-protocol that are optimized for specific network scenarios. Achieving this goal would give a flooding broadcast network the ability to adapt to and optimize itself for arbitrary traffic patterns.

In pursuit of this, Section 3.1 first describes the idealized simulated peer-to-peer network in which both the Gnutella protocol and the FBMP will be implemented. This simulated network will provide results that will allow the effectiveness of the protocols derived from the FBMP to be evaluated against the widely-used and studied Gnutella protocol.

Section 3.2 develops the FBMP itself. The aim of this meta-protocol is to provide a specification mechanism that is general enough to allow the Gnutella, s-peer and FastTrack/KaZaA protocols and a subset of the JXTA protocols to be expressed.

Finally, Section 3.3 ties these together by explaining how genetic programming will be used to derive new flooding broadcast network protocols from the FBMP that

are optimized for specific network scenarios. This can be thought of as a search for optimal protocols through the protocol-space described by the FBMP.

The end result will be a system that is able to search for new flooding broadcast network protocols that satisfy certain criteria, and to evaluate these new protocols against the existing Gnutella protocol.

3.1 The Simulation

The simulated peer-to-peer network models a flooding broadcast overlay network. This follows the lead of virtually all recent peer-to-peer research in adopting the use of an overlay network using the Internet as the underlying communication mechanism.

Overlay networks are a flexible and powerful approach and should not be considered as inferior or inefficient relative to low-level networks. Successful content-delivery overlay networks such as Akamai have already been mentioned, as well as the successes of multicast using distributed hashing networks. A case has even been made for packet-level routing using overlay networks [ABKM01] because of their greater flexibility over the Internet's low-level routing mechanisms. The point to be made here is that overlay networks are an appropriate and powerful arena in which to conduct peer-to-peer research.

The targeting of the Internet implies that the performance distribution of the machines participating in the peer-to-peer network will follow that of the Internet. Studies have shown that such a distribution on the Internet generally follows a power-law relationship [FFF99]. This in turn implies that the majority of machines participating in such a network are of a relatively low power. This conclusion is borne out by the observed participants in widely deployed peer-to-peer networks such as Napster,

Gnutella and FastTrack, which are overwhelmingly populated by end-user machines whose bandwidth bottleneck is in the ‘last hop’ to their machine over a dial-up or home broadband link.

This bias towards the ‘last hop’ bottleneck dictates the network model we will adopt. In this idealized model we assume the existence of a network backbone of infinite bandwidth, with each peer (or node) in the network connected directly to this backbone. The backbone can handle any amount of traffic, so each peer is then constrained only by the bandwidth of its connection to the backbone. Each connection that a peer maintains to another peer requires a portion of the total bandwidth of both peers’ backbone connections. All connections are modeled as symmetric for simplicity, although some real-world connections such as home broadband can have asymmetric bandwidths.

The physical model shown in Figure 3.1 shows the peers, the backbone, and the connections being maintained between the peers. The corresponding logical model in Figure 3.2 ignores the backbone and shows only the logical connections between peers.

A logical connection requires a bandwidth allotment from each of the two peers it connects, so its effective bandwidth is constrained by the lower of the two allotments available to it. In the figures shown, the link between nodes three and four is constrained to 5 units of bandwidth by node four. The remaining 40 units of bandwidth that node three has are allocated to the other connections that node three is maintaining. The link between node three and node five is constrained to 15 units by node three, so node five’s remaining 15 units of bandwidth go unused. The unused bandwidth at the unconstrained end of the link is re-allocated to other links. An

Figure 3.1: Physical network

Figure 3.2: Logical network

offline algorithm for calculating these bandwidths is given in Appendix A.

Upon joining the network, a peer broadcasts an advertisement of its presence and begins receiving replies to this broadcast informing it of the presence of other peers. As well, the peer begins working its way through its predetermined list of searches to perform. It maintains a fixed number of concurrent active searches at all times, and allots each search a window of time to receive responses before it begins the next search. Each peer also has a predetermined list of resources that it possesses, and it responds affirmatively to any queries that it receives for those resources. All peers co-operate with their neighbours by forwarding any broadcast and routed traffic that they receive from them.

Each peer maintains a collection of statistics for every other peer in the network that it is aware of. Information in this collection is gathered by examining messages

that are passed through the peer, through interactions with neighbouring peers, and through responses to messages that the peer itself has broadcast. These statistics include, for example, the shortest number of hops that messages have taken between two peers, and the number of successful responses to search queries that another peer has provided. This historical information is maintained using a sliding window of fixed size to ensure that it is always relevant.

The simulation follows a discrete-event model [BNC95], which has two attributes that are desirable in this context. The first is that the simulation operates independently of processor speed, which is important when simulating a large number of individual machines on one processor. The second is that it allows the end of the simulation to be easily detected. A negative attribute of search in a flooding broadcast network is that there is no definitive end to a given query: the querying peer has no knowledge of whether or not its query is continuing to reach new peers. In a discrete-event simulation the end of the network run is clearly established when there are no further messages to process anywhere in the network.

The simulation generates a number of statistics to aid in analyzing the performance of the network. These statistics include:

- The total number of messages routed through the network, broken down into connection and disconnection requests, arrival announcements and responses, and search queries and search responses.
- The number of successful search responses, both inside and outside of the fixed-size window following the search request.
- The total number of messages dropped because they were unroutable or because

the connection they were queued on was removed.

- The total length of the simulation.
- The average time taken and average number of peers traversed by the successful search responses.
- The small-world statistics of the network (the characteristic path length and clustering coefficient).

The purpose of the simulation is to allow different network protocols to be compared based on the performance of the network under their control, and these statistics allow such comparisons to be made. A key comparison between protocols is made by the fitness function in the genetic programming implementation, which is described in Section 3.3. A second comparison is provided by the performance graphs given in Chapter 4.

The simulation allows the network to be governed by one of two network protocols: the Gnutella protocol, described in Chapter 2, or an instance of the FBMP, described in the next section.

3.2 The Flooding Broadcast Meta-Protocol

The purpose of the FBMP is to provide a general specification mechanism that allows a wide range of specific flooding broadcast network protocols to be expressed. The range of the meta-protocol presented here encompasses the Gnutella protocol, the s-peer protocol, the FastTrack/KaZaA protocol, and a subset of the JXTA protocol. The aim of specifying such a meta-protocol is to allow a search mechanism to explore the protocol space that these protocols exist in and to arrive at specific protocols that

are optimal for specific network scenarios. This section describes the construction of the meta-protocol.

These degrees of freedom are achieved by representing each instance of the meta-protocol with two expressions. The `CONN` expression is evaluated to specify the number of connections for that peer to maintain, and the `RANK` expression is then evaluated for each existing or potential connection. The connections are ordered based on the rating the `RANK` expression gives them, and the top-ranked ones are used to fill the quota specified by the `CONN` expression.

The use of these two expressions necessarily restricts the resulting protocols to only two degrees of freedom, but this is justified by observing that these two degrees are the primary mechanisms by which most flooding broadcast protocols operate. For example: Gnutella implementations typically allow the peer administrator to set the number of connections and randomly choose the peers to connect to; s-peer hard-codes the number of connections and favours topologically-distant peers to connect to; FastTrack varies the number of connections proportionally with the bandwidth of the peer and chooses new connections randomly. A peer conceivably could have any number of additional degrees of freedom (such as, for example, the the number of neighbours it propagates broadcasts to; *c.f.* the ‘reduced branching factor’ discussion in Section 2.2.3), but the `CONN` and `RANK` expressions provide a minimal grammar that is sufficiently expressive to describe most of the flooding broadcast protocols in use today.

Both expressions make use of the standard operators `multiply`, `protdivide` (protected divide, where divide-by-zero returns 1), `add`, `subtract` and `random0to1` (which returns a random number between 0 and 1). The `RANK` expressions can also use `if`

<code>numberOfNeighbours</code>	The number of connections the peer currently has.
<code>averageMaxQueueSize</code>	The average over all of the peer's connections of the maximum number of queued messages each connection has experienced.
<code>maxQueueSize</code>	The maximum queue size of any connection the peer has.
<code>nodeBandwidth</code>	The overall bandwidth for a peer, in kilobits per second.
<code>maxSocketBandwidth</code>	The maximum bandwidth, in kilobits per second, of any connection the peer has had.
<code>totalTraffic</code>	The total amount of traffic, in bits, that the peer has processed.
<code>currentConnectionLimit</code>	The current connection limit. This is set initially to a fixed value and is then modified by subsequent evaluations of the <code>CONN</code> expression.

Table 3.1: `CONN` Expression Terminals, drawn from information gathered locally by the peer.

(which returns its second or third arguments depending on whether its first argument is positive or negative) and `greater` (which returns 1 or -1 depending on whether its first argument is greater than its second argument).

Both expressions also draw from a set of terminals representing information that the peer is able to gather locally from its observations of its immediate neighbours and of the traffic it has received. The terminals available to a `CONN` expression are given in Table 3.1. In the simulation the `CONN` expression is evaluated by a peer once every 20 seconds to provide an updated number of connections to maintain, and the terminals represent information available locally to that peer. The terminals available to a `RANK` expression are given in Table 3.2. Again every 20 seconds, each peer evaluates the `RANK` expression for every other peer that it is aware of in the network. The terminals represent information observed locally about the peer being ranked.

A representation of the Gnutella and s-peer protocols in this syntax is illustrated

<code>distance</code>	The shortest known distance, in hops, to the given peer.
<code>isNeighbour</code>	This is -1 or 1 depending on whether a connection exists between this peer and the given peer.
<code>numberOfResources</code>	The number of resources the other peer possesses. A new peer joining the network broadcasts this information, and existing peers respond with the number of resources they possess.
<code>searchHits</code>	The number of search query responses the other peer has provided.
<code>timedOutSearchHits</code>	The number of search query responses returned after this peer had expired the search.
<code>broadcastTrafficGenerated</code>	The amount of broadcast traffic (in bits) the other peer has generated.
<code>totalTraffic</code>	If a connection exists between the two peers then this is the total amount of traffic (in bits) that the peer has passed on; otherwise it is 0.
<code>originator</code>	For connected peers, this is -1 or 1 depending on whether this peer or the other peer originated the connection. Otherwise, it is 0.
<code>bandwidth</code>	The current bandwidth, in kilobits per second, of the connection between the two peers, or 0 if they are not connected.
<code>queueSize</code>	The current number of messages queued on the connection, or 0 if they are not connected.
<code>maxQueueSize</code>	The maximum number of messages queued on the connection, or 0 if they aren't connected.
<code>minBandwidth</code>	The minimum bandwidth of the connection, in kilobits per second, or 0 if there is no connection.
<code>maxBandwidth</code>	The maximum bandwidth of the connection, in kilobits per second, or 0 if there is no connection.
<code>lengthOfConnection</code>	The length of time, in milliseconds, that the peers have been connected, or 0 if they are not connected.
<code>minResponsiveness</code>	The minimum round-trip time, in milliseconds, of messages between the two peers.
<code>maxResponsiveness</code>	The maximum round-trip time, in milliseconds, of messages between the two peers.

Table 3.2: RANK Expression Terminals, drawn from information gathered locally about all other known peers in the network.

in Figure 3.3. For the Gnutella protocol, the **CONN** expression indicates that each peer aims to maintain (in this case) seven connections to other peers, and the **RANK** expression indicates that each peer favours its existing connections over other connections. In other words, a peer that has just joined the network begins by ranking all potential connections equally, so its initial choice of connections is effectively random. However, once a peer has obtained seven connections it will rank those seven connections higher than any other potential connections, effectively maintaining those existing connections and not exchanging them for new ones.

The s-peer protocol is similar in that it defaults to a fixed number (five) of connections. It also shares Gnutella's behaviour in favouring existing connections over disconnecting and obtaining new connections. However, it has the additional behaviour that when it is obtaining new connections it favours connections that are topologically far away. The **RANK** expression encapsulates this by summing the **totalTraffic** and **distance** terms. The **totalTraffic** term returns the number of bits sent by the peer being ranked if the two peers are connected, or zero if they are not. This traffic value will be greater than the distance between the two peers, so if the peers are connected, the **RANK** expression will be dominated by the **totalTraffic** term and the existing connection will be ranked highly. If the peers are not connected, however, the **RANK** expression will evaluate to the distance between the peers, effectively ranking topologically-distant peers more highly.

The FastTrack/KaZaA protocol uses supernodes, so a representation in this syntax would involve a connection limit proportional to the bandwidth of the peer.

There is no global clock in the network, so any timing information in the given terminals is based on multiple readings from the same clock. To illustrate, consider

Figure 3.3: The 7-neighbour Gnutella protocol (left) and the s-peer protocol as expression trees.

the `minResponsiveness` term. As a broadcast message propagates outwards, each node that forwards it adds a timestamp to the stack of timestamps on the message. If a node replies to the message, the response traces the reverse route back to the original sender and each node along the route extracts its own timestamp from the reply to gather timing information about the responding node. In this way nodes can establish latency measurements between themselves and other nodes in the network without relying on any global timing information.

The FBMP gives rise to a huge space of possible flooding broadcast network protocols. To search for worthwhile protocols within this space, we employ the genetic programming approach described in the following section.

3.3 Genetic Programming

We have so far developed the FBMP which allows us to describe a broad class of arbitrary flooding broadcast network protocols, and a simulated peer-to-peer network in which these protocols can be implemented and tested. Genetic programming provides the final link: the automatic generation of new protocols from the FBMP.

$$fitness = searches_{successful} + \frac{1}{2}searches_{timed-out}$$

Figure 3.4: Fitness Function

The genetic programming search for new protocols proceeds as described in Section 2.3. The **CONN** and **RANK** expressions and their set of terminals and operators provide the vocabulary, and the expressions are manipulated with the given crossover and mutation operators.

Each new protocol (represented by a pair of **CONN** and **RANK** expressions) generated by the genetic programming engine is evaluated in the simulated peer-to-peer network by applying it to the same set of peers, searches and resources as each of the other protocols, and then establishing its fitness relative to all the other protocols by ranking it according to the resulting number of successful searches.

The fitness function used to achieve this ranking is fundamental to all the genetic programming results obtained this thesis, and its choice is not taken lightly. To the best of our knowledge there are no other studies using genetic programming to derive new peer-to-peer network protocols, so we have no prior work to inform our choice of fitness function. The goal of this thesis is to develop a technique for generating optimal flooding broadcast protocols for given network scenarios, and ‘optimal’ in this context means simply that the protocol maximizes the chosen fitness function.

The actual fitness function used is given in Figure 3.4, where $searches_{successful}$ is the number of successful search responses received within the five second window following the search request, and $searches_{timed-out}$ is the number of successful search responses received outside of this window. A higher score is better. Any number of other fitness functions are possible, and a different fitness function applied to the

same network scenarios would likely give rise to an entirely different set of network protocols. The fitness function is held constant in this thesis so that the only variable affecting the evolved network protocols is the specific network scenario that the protocols arise from.

Genetic programming in the context of deriving new peer-to-peer network protocols has two desirable attributes. The first is that the new protocols produced can be easily examined to determine how they work and to gain insight into the domain of flooding broadcast network protocols in general. The second desirable attribute is that the generated protocols are relatively free of human bias and are empirically optimal within the given network scenario. (‘Optimal’ in this case means simply that the given protocol maximizes the fitness function described above.) The importance of this cannot be overstated: all of the network protocols (flooding broadcast and otherwise) described in Chapter 2—while displaying great insight and enjoying much practical success—have no claim to ‘optimality’ beyond the best guesses and hand-tuning of their creators. Genetic programming as employed here provides us with a relatively unbiased source of creativity and, with high probability, a network protocol that is optimal within the given network scenario.

The experimental results presented in Chapter 4 were obtained using a modified form of the freely-available `lil-gp` genetic programming package from the University of Michigan [PZ98]. The settings used are given in Appendix B.

3.4 Summary

In this chapter a generalized flooding broadcast meta-protocol (FBMP) was presented that allows a wide range of flooding broadcast network protocols to be described. A

simulated peer-to-peer network was also presented which is capable of simulating the widely-used Gnutella protocol as well as protocols derived from the FBMP. Finally, genetic programming was proposed as a method to automatically generate new flooding broadcast network protocols and to guide a search through the space of possible protocols in pursuit of protocols that are optimal for specific network scenarios. The results of these searches are presented in the next chapter.

Chapter 4

Results

To establish the viability of using genetic programming to search for new flooding broadcast network protocols, the simulated peer-to-peer network described in the previous chapter was used to generate optimal protocols from the FBMP for a variety of specific network scenarios. This chapter describes the results of these simulations.

The scenarios themselves are described in Section 4.1. A number of different graphs and diagrams are used to present the experimental results, and these are described in Section 4.2. The scenarios are divided into a number of groups, and the results for each of the groups are presented in the remaining sections. The chapter concludes with a discussion of the performance of the simulation.

The analysis of these results is delayed until Chapter 5.

4.1 The Scenarios

The scenarios are intended to be a representative sampling of realistic peer interactions and traffic flows in a typical peer-to-peer network. The scenarios include such variations as differing peer bandwidths to simulate the discrepancies between dial-up and broadband users, and differing ‘resource pools’ to simulate peers seeking and

providing disjoint sets of resources. The scenarios cover a broad range of differing peer bandwidths and types of traffic, but are not exhaustive; any number of additional scenarios could be created and would likely give rise to new optimal network protocols.

Scenario A is intended to contrast with Scenario B. In each of Scenario A and B, the bandwidths of the peers are varied in seven sub-scenarios from 40 kbps (56k modem speed) up to 500 kbps ($\frac{2}{3}$ of a home broadband connection). In Scenario A each node draws from the same set of resources. In Scenario B, each node draws exclusively from one of two distinct sets of resources. The results for Scenario A are presented in Figures 4.1 through 4.21. The results for Scenario B are presented in Figures 4.22 through 4.42.

Scenario C represents networks containing multiple distinct subgroups. In all presented scenarios 70% of the peers are low bandwidth (40 kbps) and 30% of the peers are high bandwidth (800 kbps). The number of distinct subgroups ranges from one in Scenario C-1 up to five in Scenario C-5. The results for Scenario C are presented in Figures 4.43 through 4.57.

Scenario D is similar to Scenario C-1, except that the 40 kbps / 800 kbps disparity between peer bandwidths in Scenario C-1 is reduced in Scenario D to 200 kbps and 600 kbps. The peers in Scenario E draw from a single set of resources, but possess one of three bandwidths: 40 kbps, 800 kbps or 12000 kbps. Scenario F is intended to show that the simulation is scale-free and behaves independently of the number of peers in the network. It contains 50 nodes of similar bandwidth, where each draws from one of two pools of resources.

A breakdown of these scenarios is given in Table 4.1. Each peer in the network

	A	B	C	D	E	F
Sub-scenarios	7	7	5	1	1	1
Peers	30	30	30	30	30	50
Peer Bandwidths (kbps)	40–500	40–500	40/800	200/600	40/800/12000	200
Resource Pools	1	2	1–5	1	1	2
Resources per Peer	50	50	50	50	50	50
Searches per Peer	200	200	200	200	200	200
Figures	4.1–4.21	4.22–4.42	4.43–4.57	4.58–4.60	4.61–4.63	4.64–4.66

Table 4.1: Breakdown of Scenarios

has a fixed maximum bandwidth, as described in Section 3.1. These bandwidths correspond roughly to the real world as follows: a 56k modem connection is roughly 40 kbps, a home broadband connection is roughly 800 kbps, and a T1 line is roughly 12000 kbps.

Each peer also has a fixed set of resources that it possesses, and a fixed set of searches that it performs. These resources are drawn from a pool of resources shared by other peers. The ‘Resource Pools’ property refers to the number of distinct pools that the peers’ resources are drawn from. In Scenario A, for example, all peers draw from the same set of resources. In Scenario B, each draws exclusively from one of the two pools of resources.

Each scenario was re-run a number of times with the two different protocols in place. In the case of the Gnutella protocol, the results presented are the average of 20 runs. For the FBMP results, the ‘fittest’ evolved protocol is presented. In all cases the FBMP runs were re-done either two or three times, and in all cases but one (Scenario C-3, discussed below) they converged to the same result.

4.2 Presentation of Results

In the following section, each scenario (or sub-scenario) is presented in three parts. First, a diagram of the final network topology for the ‘most fit’ evolved FBMP protocol is given. This shows the state of the network at the point where all peers have finished performing their assigned searches and there are no more messages propagating through the network. Peers are identified in two ways: a larger diameter indicates a higher bandwidth, and a different grey-shade indicates that the peer draws its resources and searches from a different pool. For example, Figure 4.1 shows the final network topology for the optimal FBMP-derived protocol in scenario A-1. This protocol organized the peers into a ring where each peer had a degree of two. All peers were the same speed (hence all peers in the diagram have the same diameter), and drew from the same set of resources (hence they all share the same colour).

The fitness function used in the simulations dictated that the evolved protocols were rewarded for the number of successful searches that occurred in networks under their control. The strategy employed by the optimal protocol is often immediately apparent from this first diagram. For example, a protocol may enforce a fixed number of connections per peer, or it may vary this according to some formula. As well, the protocol may aim to organize the network into a hierarchical structure, or it may fragment the peers into groups along resource-pool lines.

The second diagram shows the **CONN** and **RANK** expression trees for the optimal FBMP protocol. These expressions draw from the set of terminals given in Tables 3.1 and 3.2. For example, Figure 4.2 gives the expression trees used to generate the network in the first diagram. The **CONN** expression corresponds to ‘**random0to1 - -2.0**’. This evaluates to a value in the range $[2, 3)$ which, when the absolute value

and floor operations are applied, gives the degree-two result we see in the topology diagram.

The final diagram gives a number of graphs measuring the performance of the two protocols. Each graph shows the results of the Gnutella protocol evaluated for each of degree 3 through degree 20, combined with the results for the most fit FBMP protocol. Each graph thus contains 18 results for the Gnutella protocol (degrees 3 through 20), and one result for the best evolved FBMP protocol. In the first two graphs, the FBMP result is shown as a separate bar to the right of the graph. In all other cases, the FBMP result is shown as a dotted horizontal line.

The *Successful+Timed-Out Messages* graph gives the successful searches achieved by each protocol as the lower bar, with the ‘timed-out’ searches (successful search responses received after the search window has closed) shown as the upper bar. The fitness function that the protocol sought to maximize was explicitly stated in terms of the number of successful searches, so this graph gives the ‘fitness’ of the evolved protocol and allows it to be compared against the performance of the Gnutella protocol.

The *Total Messages* graph shows the total number of messages processed by the network as a vertical bar, with subcomponents for three groups of messages: the *connection* messages, which are the initial connection and disconnection requests and arrival broadcast and response messages; the *search* messages, which are the broadcast queries for specific resources; and the *search response* messages, which are the successful responses to the search messages. The intent of this graph is to show the relative efficiency of the networks: a protocol that generates twice as many overall messages as another protocol but that achieves the same number of successful searches

is certainly much less efficient. This graph also shows the effects of increasing the number of connections in the Gnutella protocol.

The *Average Hops* graph gives the average number of hops between peers taken by each successful (or timed-out) search response. It indicates the proximity of related peers in the network, and also shows the effects of increasing the number of connections in the Gnutella protocol.

The *Average Search Time* graph shows the average amount of time between each peer initiating a search and receiving a successful (or timed-out) response for it. The intent of this graph is to show the performance of the network in terms that a user of the system would observe. The difference between an average search time of 5 seconds and 50 seconds, for example, is the difference between a usable and an unusable system in the eyes of a user.

The *Small World Factor* graph shows the ‘proximity ratio’ [Wal99] of the networks, defined as the ratio of the clustering coefficient to the characteristic path length. These graphs are presented without a scale as their only purpose is to compare the ‘small-worldliness’ of a protocol’s results with the ‘small-worldliness’ of a random graph with the same number of vertices and edges. A higher value in the graph simply indicates that the graph exhibits a more small-world-like behaviour: that is, a greater degree of clustering while retaining a relatively small diameter. Comparisons in this graph should be made carefully: a value from the ‘Gnutella’ series can be compared only against the corresponding value from the ‘random’ series for the same number of connections. The intent of this graph is to indicate when evolved FBMP protocols result in networks with or without small-world behaviour.

4.3 Results

This section presents the charts and graphs detailing the results for all scenarios.

Figure 4.1: Scenario A-1 evolved network topology. All peers are 40 kbps.

Figure 4.2: Scenario A-1 evolved protocol expressions. The `totalTraffic` term favours existing neighbours, but new connections are ranked with `-maxResponsiveness` which favours other close connections.

Figure 4.3: Scenario A-1 performance graphs.

Figure 4.4: Scenario A-2 evolved network topology. All peers are 70 kbps.

Figure 4.5: Scenario A-2 evolved protocol expressions. The **totalTraffic** term dominates for neighbouring connections, so this protocol instructs peers to retain their neighbours. The term returns 0 for non-neighbours however, so new connections are ranked according to **maxResponsiveness** which favours topologically distant peers.

Figure 4.6: Scenario A-2 performance graphs.

Figure 4.7: Scenario A-3 evolved network topology. All peers are 100 kbps.

Figure 4.8: Scenario A-3 evolved protocol expressions. The `lengthOfConnection` term retains existing neighbours, but the `distance` terms dominate for non-neighbours so new connections favour topologically distant peers.

Figure 4.9: Scenario A-3 performance graphs

Figure 4.10: Scenario A-4 evolved network topology. All peers are 200 kbps.

Figure 4.11: Scenario A-4 evolved protocol expressions. The **totalTraffic** term dominates for existing connections so neighbours are favoured, but the **distance** terms dominate otherwise so new connections are with topologically distant peers.

Figure 4.12: Scenario A-4 performance graphs

Figure 4.13: Scenario A-5 evolved network topology. All peers are 300 kbps.

Figure 4.14: Scenario A-5 evolved protocol expressions. The initial connection limit for each peer is 10 connections, so the `currentConnectionLimit` term simply maintains this. All peers have the same number of resources, so the `totalTraffic` term is the only meaningful one in the `RANK` expression. This term is non-zero for neighbours and zero for non-neighbours, so this protocol is equivalent to the 10-neighbour Gnutella protocol.

Figure 4.15: Scenario A-5 performance graphs

Figure 4.16: Scenario A-6 evolved network topology. All peers are 400 kbps.

Figure 4.17: Scenario A-6 evolved protocol expressions. The **isNeighbour** term favours existing neighbours while the rest of the expression bases new connections on low broadcast traffic and high search hits. This protocol is essentially equivalent to the 10-neighbour Gnutella protocol.

Figure 4.18: Scenario A-6 performance graphs

Figure 4.19: Scenario A-7 evolved network topology. All peers are 500 kbps.

Figure 4.20: Scenario A-7 evolved protocol expressions. The CONN expression reduces to simply `currentConnectionLimit`, so this protocol is exactly the 10-neighbour Gnutella protocol.

Figure 4.21: Scenario A-7 performance graphs. Small differences between the results for the FBMP protocol and the results for the 10-neighbour Gnutella protocol exist because of stochastic processes within the simulation and because the Gnutella protocol measurements are the average of 20 runs while the FBMP measurements are derived from only a single run.

Figure 4.22: Scenario B-1 evolved network topology. All peers are 40 kbps.

Figure 4.23: Scenario B-1 evolved protocol expressions. The **totalTraffic** and **bandwidth** terms favour existing neighbours but the **searchHits** term negates the ranking if the peer has not provided any successful search responses, producing the observed separation by resource pools. New connections are ranked by the **maxResponsiveness** term, which favours topologically distant peers.

Figure 4.24: Scenario B-1 performance graphs

Figure 4.25: Scenario B-2 evolved network topology. All peers are 70 kbps.

Figure 4.26: Scenario B-2 evolved protocol expressions. The `maxQueueSize` and `totalTraffic` terms are 0 for non-neighbours, so the protocol essentially maintains existing neighbours. New connections are ranked by the `distance` term, favouring topologically distant peers.

Figure 4.27: Scenario B-2 performance graphs

Figure 4.28: Scenario B-3 evolved network topology. All peers are 100 kbps.

Figure 4.29: Scenario B-3 evolved protocol expressions. The left branch of the **RANK** expression is dominated by the **totalTraffic** and **queueSize** terms which favour existing neighbours. The left branch is zero for non-neighbours however, so the **-distance/minResponsiveness** term in the right branch favours peers which have large minimum responsivenesses. (These large response times indicate topologically distant peers).

Figure 4.30: Scenario B-3 performance graphs

Figure 4.31: Scenario B-4 evolved network topology. All peers are 200 kbps.

Figure 4.32: Scenario B-4 evolved protocol expressions. The `queueSize` term favours existing neighbours. New connections are ranked by `distance/broadcastTrafficGenerated`, favouring topologically distant peers.

Figure 4.33: Scenario B-4 performance graphs

Figure 4.34: Scenario B-5 evolved network topology. All peers are 300 kbps.

Figure 4.35: Scenario B-5 evolved protocol expressions. The `CONN` expression reduces to `currentConnectionLimit` and the `RANK` expression is essentially `isNeighbour`, so this protocol is equivalent to the 10-neighbour Gnutella protocol.

Figure 4.36: Scenario B-5 performance graphs. Small differences between the results for the FBMP protocol and the results for the 10-neighbour Gnutella protocol exist because of stochastic processes within the simulation and because the Gnutella protocol measurements are the average of 20 runs while the FBMP measurements are derived from only a single run.

Figure 4.37: Scenario B-6 evolved network topology. All peers are 400 kbps.

Figure 4.38: Scenario B-6 evolved protocol expressions. The `isNeighbour * bandwidth` expression dominates, causing the protocol to favour neighbours. New connections are chosen essentially randomly, so this protocol is equivalent to the 10-neighbour Gnutella protocol.

Figure 4.39: Scenario B-6 performance graphs

Figure 4.40: Scenario B-7 evolved network topology. All peers are 500 kbps.

Figure 4.41: Scenario B-7 evolved protocol expressions. The **isNeighbour** term dominates the **RANK** expression, so existing neighbours are maintained. The **numberOfResources** term is constant and the **maxBandwidth** term is zero for non-neighbours, so new connections are chosen randomly by the **random0to1** term. This is equivalent to the 10-neighbour Gnutella protocol.

Figure 4.42: Scenario B-7 performance graphs

Figure 4.43: Scenario C-1 evolved network topology. Peers are either 40 kbps or 800 kbps and draw from the same set of resources.

Figure 4.44: Scenario C-1 evolved protocol expressions. The **CONN** expression is roughly $\text{nodeBandwidth}/\text{maxQueueSize}$, which evaluates to 1 or less for 40 kbps peers with significant queues on their connections but which evaluates to a much higher value for 800 kbps peers. This forms the cluster of high-bandwidth ‘supernodes’. The **totalTraffic** terms in the **RANK** expression favour existing connections.

Figure 4.45: Scenario C-1 performance graphs

Figure 4.46: Scenario C-2 evolved network topology. Peers are either 40 kbps or 800 kbps and draw from two distinct pools of resources.

Figure 4.47: Scenario C-2 evolved protocol expressions. The CONN expression is roughly $\text{numberOfNeighbours} * (\text{numberOfNeighbours} - \text{currentConnectionLimit})$, which allows one connection limit to run away while the others remain at 1. (If the CONN expression evaluates to zero, a one is substituted instead.) The bandwidth and maxQueueSize terms in the RANK expression favour existing connections.

Figure 4.48: Scenario C-2 performance graphs

Figure 4.49: Scenario C-3 evolved network topology. Peers are either 40 kbps or 800 kbps and draw from three distinct pools of resources. The evolved protocols for Scenarios C-1 and C-2 show supernode behaviour, while the protocols for Scenarios C-4 and C-5 show group-separation. Scenario C-3 seems to occupy the transitional zone between these two types of protocols, and the experimental evolutionary runs consistently converged on one or the other strategy. The supernode strategy achieved a slightly better fitness rating, and is the one presented.

Figure 4.50: Scenario C-3 evolved protocol expressions. The CONN expression makes good use of an if statement to separate the capabilities of the two peer bandwidths. The `averageQueueSize > currentConnectionLimit` term separates the capabilities, assigning the 40 kbps peers a connection limit of one and the 800 kbps a much higher limit. This results in the observed ‘supernode’ behaviour. The `maxBandwidth` term in the RANK expression favours existing connections.

Figure 4.51: Scenario C-3 performance graphs

Figure 4.52: Scenario C-4 evolved network topology. Peers are either 40 kbps or 800 kbps and draw from four distinct pools of resources.

Figure 4.53: Scenario C-4 evolved protocol expressions. The `timedOutSearchHits` terms in the `RANK` expression favour peers that draw from the same resource pool. It simultaneously favours topologically distant peers by only considering ‘timed-out’ messages.

Figure 4.54: Scenario C-4 performance graphs

Figure 4.55: Scenario C-5 evolved network topology. Peers are either 40 kbps or 800 kbps and draw from five distinct pools of resources.

Figure 4.56: Scenario C-5 evolved protocol expressions. The **searchHits** term favours peers drawing from the same resource pool, but if there are no search hits then the **maxResponsiveness** terms favour topologically distant peers.

Figure 4.57: Scenario C-5 performance graphs

Figure 4.58: Scenario D evolved network topology. Nodes are either 200 kbps or 600 kbps.

Figure 4.59: Scenario D evolved protocol expressions. The **CONN** expression reduces to $\text{nodeBandwidth} / \text{averageQueueSize}$, which controls the node degree with a feedback-limiting approach: if the peer is overloaded, the **averageQueueSize** increases which reduces the connection limit and sheds some of the load. The **RANK** expression simply favours existing connections.

Figure 4.60: Scenario D performance graphs

Figure 4.61: Scenario E evolved network topology. Nodes are either 40 kbps, 800 kbps or 12000 kbps.

Figure 4.62: Scenario E evolved protocol expressions. The **CONN** expression follows the familiar pattern of varying the connection limit proportionally with the bandwidth (through **maxSocketBandwidth**) and inversely with the average queue size (through **averageQueueSize**). The **RANK** expression is dominated by **totalTraffic**⁴ which favours existing connections.

Figure 4.63: Scenario E performance graphs

Figure 4.64: Scenario F evolved network topology. All nodes are 200 kbps.

Figure 4.65: Scenario F evolved protocol expressions. The **CONN** expression is fixed, while the **RANK** expression combines a preference for topologically distant peers (**distance**) with preferences for similar resource pools (**searchHits**) and existing connections(**bandwidth**). (The **numberOfResources** term is constant.)

Figure 4.66: Scenario F performance graphs

4.4 Performance

The benchmark machine used was a 1 GHz AMD system with 512 MB of memory. This machine took 12 hours to run a full suite of 30-node Gnutella simulations consisting of 20 network runs for each of 18 node degrees, for a total of 360 simulations. The machine also took 12 hours to process roughly 50 generations of a population of 200 individual FBMP protocols. The evolutionary search cached the results for FBMP protocols that had already been processed and terminated un-promising protocols that exceeded limits on message queues, so the number of network runs in the course of this search was roughly comparable to the number of Gnutella runs that could be completed in the same amount of time.

The decision to limit the simulated networks to 30 peers reflects a balance between a number of factors. The obvious factor is the desire to make the networks as large as possible to more realistically model real-world peer-to-peer networks. The opposing factor is the limited processing power available. In the end, 30 peers (and the 12-hours per run entailed) was considered an upper limit on the number of peers that would give relevant results but would still allow experimental data to be gathered in a timely manner.

4.5 Conclusion

This chapter has presented a suite of results from a variety of network scenarios illustrating both the Gnutella protocol and some optimal evolved protocols derived from the FBMP. These results are discussed in detail in the following chapter.

Chapter 5

Analysis

This chapter presents an analysis of the results described in the previous chapter. The analysis focuses on the workings of the various protocols that were shown to be optimal in the different network scenarios. The evaluation of whether the overall goals of this thesis have been met is delayed until Chapter 6.

5.1 Scenarios A and B

Scenarios A and B show a network of peers with similar bandwidths varying along two orthogonal degrees of freedom: the shared bandwidth, and the number of distinct pools of resources the peers draw from.

In this section we will first look at the performance¹ of the Gnutella protocol, and then compare the performance of the best evolved protocols against it

5.1.1 Gnutella

The Gnutella protocol exhibits some interesting properties in these scenarios. In the initial low-bandwidth scenarios A-1 and B-1 (Figures 4.3 and 4.24) the search times for

¹‘Performance’ in this context refers to the overall functioning of the network under the control of the given protocol.

the Gnutella protocol range from 25 up to 70 seconds, well outside of the five-second window for ‘successful’ searches. This is shown clearly in the *Successful+Timed-Out Messages* graphs, which show that virtually all of the searches were considered ‘timed-out’.

As the bandwidth increases, though, we start to see coherent behaviour. In scenario B-3 (Figure 4.30) with node degrees 3–4 and scenario A-4 (Figure 4.12) with node degrees 3–7 we see the search times steady for the first time below the 5-second mark, but chaotic for higher node degrees. As the bandwidth is increased further, we see that this threshold between the coherent, efficient functioning of the network and the chaotic, saturated behaviour increases until by scenarios A-6 and B-6 (Figures 4.18 and 4.39) all node degrees are functioning efficiently within the five-second search window.

This corresponds with observed behaviour in the real-world Gnutella network. As mentioned in Section 2.2.3, Clip2.com characterized these phase shifts in the Gnutella network as ‘pre-barrier’ and ‘post-barrier’, with the shift occurring at the point the average amount of traffic in the network grew to exceed the capacity of a 56k modem link [Cli00]. At this point the real-world network fractured into disjoint subnetworks bordered by the swamped peers which were no longer able to participate in the network. In the simulated networks, this barrier is clearly visible as a function of both individual bandwidth constraints on each peer and increasing overall traffic in the network due to increasing node degrees. (This increasing traffic is visible in the standard pattern repeated in each scenario in the *Total Messages* graph: as the degree of each peer increases, the total number of messages processed in the network increases linearly with it.)

A second observation about the Gnutella protocol involves the *Small World Factor* graphs. These graphs compare the small-world factors of the Gnutella and generalized Gnutella protocols against random graphs with similar numbers of edges and nodes. We see the same pattern repeated in all of the scenarios: the small-world factor for the Gnutella graphs is higher than for correspondingly complex random graphs. Again, this corresponds to observations made of real-world Gnutella networks. As mentioned in Section 2.2.3, real-world Gnutella networks exhibit strong small-world characteristics [JAB01]. Other authors have wondered about the reason for this [IRF02], and have posited that Gnutella’s discovery mechanism is necessarily biased towards peers that are topologically close. Building connections with these close peers then results in a high degree of clustering, which leads to the observed small world characteristics.

The results seen in these simulations show that the Gnutella protocol does indeed build small-world networks on its own, and that these characteristics are not an artifact of some external force such as social relationships between peer maintainers.

5.1.2 FBMP

The most obvious trend in the FBMP protocols evolved for Scenarios A and B is that the ‘optimal’ node degree varies proportionally with the bandwidth of the peers. We can see this in the network diagrams and also in the `CONN` expressions. In the 40 kbps case (Scenarios A-1 and B-1, Figures 4.1 and 4.22, respectively), the evolved protocols use degrees 2 and 3, respectively. This ranges up to `currentConnectionLimit` in Scenarios A-5 (Figure 4.13), B-5 (Figure 4.34) and above. The connection limit begins as 10, so the `currentConnectionLimit` term simply maintains this node degree. In practice the networks are able only to approach this limit, and we see average node degrees ranging roughly between 7 and 9. This relation is shown in Figure 5.1, where

Figure 5.1: Average node degrees of evolved FBMP networks for Scenarios A and B

the average node degrees have been extracted from the networks resulting from each of the ‘most fit’ FBMP protocols over two runs each through Scenarios A and B.

The significant insight that we gain from these scenarios, however, is in the methods that the protocols use to organize the networks. In Scenarios A-1 to A-4 (Figures 4.2, 4.5, 4.8 and 4.11) and B-1 to B-4 (Figures 4.23, 4.26, 4.29 and 4.32) we see that the protocols adopt Gnutella’s strategy of maintaining their existing set of connections, but they diverge significantly from Gnutella in the way that they obtain new connections. All protocols in this lower range engage in a constant process of ‘short-circuiting’ the network by favouring connections with nodes that are topologically distant.

This mirrors the method proposed by Watts and Strogatz to move from a highly-ordered ‘lattice’ network towards a random network, with the goal of arriving at a ‘small world’ network somewhere between the two extremes [WS98]. Examining the

small-world factors for the networks produced in Scenarios A-1 to A-4 (Figures 4.3, 4.6, 4.9 and 4.12) and B-1 to B-4 (Figures 4.24, 4.27, 4.30 and 4.33), we see that the evolved protocols consistently ranked at or below the factor for the correspondingly complex random graph. This shows that these evolved protocols are substantially different from the Gnutella protocol in that they do not lead to small-world networks.

The evolved behaviour is markedly different in Scenarios A-5 (Figure 4.14), B-5 (Figure 4.35) and up. The evolved protocols are equivalent to the Gnutella protocol, although each has adopted a node degree that is optimal to its context. And as expected with the use of the Gnutella protocol, we see in the *Small World Factor* graphs that the networks resulting from these protocols have significantly higher small-world factors than the correspondingly complex random graphs. This ‘small-worldliness’ is also evident in the network diagrams, which exhibit the characteristic ‘cliquishness’ of small world networks.

The division between these two different strategies is also visible in the average node degrees in Figure 5.1. The scenarios up to A-4 and B-4 (from 40 kbps to 200 kbps) in which short-circuiting was employed use a low (less than 5) node degree, while Scenarios A-5, B-5 and above (from 300 kbps to 500 kbps) in which the Gnutella protocol was employed shows a significant jump in average node degree to greater than 7.

Another emergent strategy reveals a limitation of the model we have employed, and of the Gnutella protocol itself. Separation between resource pools occurred in Scenario B-1 (Figure 4.22), but not in any of the other ‘B’ scenarios. An examination of protocols that the genetic programming engine considered and rejected in Scenario B shows numerous introductions of resource-pool separation protocols. It also shows

that these protocols were ‘less fit’ than the protocols described above, because they resulted in fewer successful searches.

The reason for this involves a diversion into the details of the simulation’s routing mechanism, which mirrors Gnutella’s. As shown in Figure 2.1, a response in a Gnutella network is routed back along exactly the same path that the original message took. Each peer along this path tracked the ID and originator of the original message, and uses this information to route the reply along the reverse path. The problem arises when one of the connections along this path is dropped—this is because a peer receiving a message destined for a connection that has already been dropped will simply drop the message. This is a recognized shortcoming of the Gnutella protocol, and is shared by products such as s-peer. At least one Gnutella implementation has attempted to remedy this by having peers revert to broadcast behaviour when they receive messages they cannot route. JXTA addresses the problem by allowing messages to contain their own routing information, and by introducing the concept of ‘peer routers’ that peers can request aid from when they cannot route a message themselves.

This facet of Gnutella’s routing embodies something of a contradiction in that the protocol was invented to service a file-sharing overlay network on the Internet in which highly-transient peers are the norm, but transient peers are the case that the Gnutella protocol doesn’t handle gracefully. Admittedly, this is mitigated by the Gnutella protocol’s bias towards maintaining existing connections and not dynamically modifying the topology of the network.

Returning to the question of why the resource-pool separation protocols were considered less-fit, we now see that ‘dynamic’ protocols that frequently drop and

add new connections suffer a fitness penalty because they result in more unroutable messages and therefore fewer successful searches. Thus the resource-pool separation seen in Scenario B-1 is significant because it was the optimal FBMP protocol despite the implicit penalty it was paying by frequently changing connections.

Two potential solutions to this problem are as follows: a first solution would involve modifying the protocol to gracefully handle unroutable messages through a mechanism similar to JXTA's. The introduction of peer routers and smarter messages, however, would add a significant amount of complexity and be a divergence from existing protocols such as Gnutella. A second solution would modify the fitness rating method by introducing some form of 'training period', where the network would be allowed to self-organize for a period of time before the gathering of fitness statistics was begun. This would allow the network to organize itself along whatever lines the protocol dictated, while not paying a penalty for the dropped messages that this rearrangement necessarily implies.

5.2 Scenario C

Scenario C illustrates a transition from a network where all peers draw from the same pool of resources to a network where the peers draw from up to five distinct pools. The peers also have a fixed division between bandwidths: 70% are 40 kbps and 30% are 800 kbps.

5.2.1 Gnutella

In all cases, we see Gnutella performing in its 'chaotic' mode. The average search times rise from roughly 25 seconds in Scenario C-1 (Figure 4.45) through 55 seconds, 80

seconds and 100 seconds in the remaining scenarios (Figures 4.48, 4.51, 4.54 and 4.57). In the *Successful+Timed-Out Messages* graphs we see that only a small percentage of the searches are completed successfully within the five-second timeout window. These results show that the Gnutella protocol is not suitable for the given network scenarios.

We also see in the *Small World Factor* graphs a confirmation of the observation we made in Scenarios A and B: the Gnutella protocol leads to network topologies with small-world characteristics.

5.2.2 FBMP

In the evolved FBMP protocols we see two distinct strategies emerge. In Scenarios C-1 and C-2 (Figures 4.43 and 4.46) where there are only one or two distinct pools of resources, we see protocols which exhibit ‘supernode’ behaviour. In Scenario C-1, this takes the form of a central cluster of high-bandwidth peers with the low-bandwidth peers maintaining a single connection into this cluster. In Scenario C-2 we see the network organized around a single high-bandwidth broadcast hub.

This contrasts with the protocols in Scenarios C-4 and C-5 (Figures 4.52 and 4.55). These protocols segregate the network into subgroups based on resource pools, and make no distinction between the bandwidths of the peers.

Scenario C-3 (Figure 4.49) occupies the transitional zone between these distinct strategies. Multiple runs of the genetic programming search converged on one or the other strategy, with the supernode cluster protocol being slightly more fit.

The insight we draw from this is that supernode behaviour is efficient when the peers being served benefit from at least a significant proportion of the traffic they receive. In Scenarios C-4 and C-5 where 75% to 80% of the traffic in the network was

not relevant to a given peer, it proved more efficient to balkanize the network into subgroups that interacted only with other like peers.

5.3 Scenarios D, E and F

The last three scenarios are variations of earlier scenarios. In all three cases we observe that the Gnutella protocol performs passably well, but is not overly efficient in any of the scenarios. We also observe that the Gnutella protocol continues to result in networks with high small-world characteristics.

Scenario D is similar to Scenario C-1 except that the disparity in bandwidths between the peers is reduced. The FBMP protocol that emerges (Figure 4.59) is identical to that in Scenario C-1 (Figure 4.44), and follows the now-standard pattern of varying the connection limit proportionally with the peer's bandwidth and inversely with the queue sizes of the peer's connections. This approach ensures that high-bandwidth peers have correspondingly high connection limits, but also employs the feedback mechanism of using the connection queue sizes to indicate whether the peer is overloaded or not. This mechanism causes an overloaded peer to shed connections until its queue sizes are brought down.

Scenario E is similar again, except that each peer has one of three distinct bandwidths. We see the emergence (Figure 4.62) of the same protocol discussed above, which has the effect in this scenario of creating three 'tiers' of peers: an inner, very high bandwidth hub, a medium-bandwidth ring around this, and the remaining low-bandwidth peers loosely connected on the periphery.

Scenario F is intended to show that the results seen so far are in no way dependent on the number of peers in the network. We have 50 peers of equal bandwidth that

draw from two resource pools, so this scenario is similar to Scenario B-4. Unlike the results seen in Scenario B-4 (Figure 4.31) however, we see the emergence (Figure 4.64) of a resource-separation protocol as optimal.

We observed in Scenario B-1 that a resource-pool-separation protocol was optimal in that case, despite the implicit penalty it was paying for dynamically changing connections. The optimal protocol observed for Scenario F leads us to believe that resource-pool-separation strategies become increasingly important as the size of the network increases and small-world style short-circuiting becomes less effective.

5.4 Summary

In this chapter we have made the following observations:

- The Gnutella protocol exhibits two distinct behaviours characterized by search times that differ by an order of magnitude: coherent and efficient operation when the peers' bandwidth is sufficient, and chaotic, inefficient operation when the peers are bandwidth-constrained.
- The Gnutella protocol arose numerous times as an optimal FBMP protocol. This leads us to conclude that when the Gnutella protocol is working efficiently, it is often optimal.
- The Gnutella protocol is not able to take advantage of disparities between peer bandwidths.
- The Gnutella protocol on its own results in networks with high small-world factors.

- The ‘optimal’ node degree utilized by the evolved FBMP protocols varies proportionally with the bandwidth of the peers.
- In homogeneous networks where all peers were drawing from the same resource pools, two distinct FBMP protocols arose: in the low-bandwidth cases that resulted in chaotic Gnutella behaviour, we see the adoption of a low node degree, ‘short-circuiting’ protocol similar to the s-peer protocol that explicitly creates non-small-world networks. In the high- bandwidth cases, we see the emergence of the Gnutella protocol itself as optimal.
- Resource-pool separation strategies and other dynamic protocols are penalized by the Gnutella protocol’s inability to handle unroutable messages, although in many cases they are still optimal despite this penalty.
- Supernode strategies are also very effective, and are characterized by protocols that vary the connection limit of the peer proportionally with the bandwidth available to the peer and inverse-proportionally to the average or maximum size of the queues on the peer’s connections. This allows peers to take advantage of high bandwidths, and employs a ‘throttling’ feedback mechanism to shed connections if a peer becomes overloaded.
- The Gnutella protocol is optimal in scenarios in which the peers are not bandwidth-constrained and the resources being sought are fairly homogeneous.

We conclude by noting that in all cases the search times achieved by the evolved FBMP protocols are very reasonable (they are mostly within the five-second timeout window), even in the cases where the Gnutella protocol is highly inefficient.

5.5 Conclusion

In this chapter we have presented an analysis of the experimental results given in Chapter 4. The primary focus of this analysis was to characterize the optimal flooding broadcast network protocols for the various network scenarios that were used.

The following chapter evaluates the success of the FBMP approach in meeting the overall goal of this thesis, and discusses possible future work.

Chapter 6

Conclusion

In this chapter, we first confirm that our initial goals have been met. We then itemize the contributions made by this thesis, and conclude with some suggested avenues for future work.

6.1 Confirmation of Approach

The main goal of this thesis is to establish that genetic programming is a viable method for deriving flooding broadcast network protocols that are optimal for given network scenarios.

From the results and analysis presented in the previous chapters, we see that in all cases multiple applications of this approach have converged on identical, empirically optimal network protocols. As well, we have seen that the optimal protocols derived for related scenarios are themselves related. For example, Scenarios A-1 to A-4 and B-1 to B-4 all arrive at optimal protocols that maintain a fixed number of neighbours and that favour topologically distant peers for new connections. These strategies avoid small-world behaviour by constantly ‘short-circuiting’ the network. Similarly, Scenarios A-5 to A-7 and B-5 to B-7 all arrive at the Gnutella protocol as their optimal

protocol. These independent arrivals at behaviourally-identical optimal protocols reinforce the success of the method used and show that it is robust in the face of changing network scenarios. We therefore conclude that the primary goal of the thesis has been achieved.

We have also seen good success in the network protocols that the genetic programming system has produced. In some network scenarios the Gnutella protocol has emerged as optimal, while in others the optimal strategies have been different. In all cases, however, and including the scenarios in which the Gnutella protocol struggled, the evolved FBMP protocols always showed coherent and sensible behaviour. The evolved protocols always resulted in high numbers of successful searches, and often achieved average search times an order of magnitude lower than Gnutella's. This comparison against a real-world, deployed protocol such as Gnutella confirms the effectiveness of the approach used in this thesis.

The approach has a number of drawbacks, however. One drawback is that each peer in the network utilizing the FBMP protocol must maintain information about every other peer in the network that it is aware of. The amount of information maintained is small (just enough to satisfy all of the terminals in Table 3.2), but in a very large network this requirement could become onerous. This space requirement could be capped with the use of a fixed-size cache, but the effect of such a decision on the functioning of the evolved protocols is unknown.

A second limitation of the approach used in this thesis is that the genetic programming search requires the 'pre-processing' of a representative sample of traffic to arrive at an optimal network protocol that would then be hard-coded into the live

network. This traffic sample may not be available, may be difficult to characterize, or the network itself may vary considerably over time in the traffic patterns it experiences.

Overall, however, the approach described in this thesis for deriving new flooding broadcast network protocols has been shown to be viable and the protocols themselves to be effective and efficient.

6.2 Contributions

The contributions of this thesis are as follows:

- The use of genetic programming to search through the protocol space of the FBMP for flooding broadcast network protocols is shown to be a viable and effective method for generating protocols that are optimal in specific network scenarios.
- A variety of specific flooding broadcast network protocols are described, along with the network scenarios in which they are optimal. The insight gained from the examination of these optimal protocols may provide guidance to designers of future flooding broadcast network protocols.
- An offline algorithm for determining bandwidths in a ‘last hop’ network is presented in Appendix A.
- The Gnutella protocol itself is shown to give rise to small-world networks. This removes any speculation that factors external to the protocol cause the small-world characteristics observed in real Gnutella networks.

6.3 Future Work

There are a number of avenues of approach that future work in this area could take. This section describes the most interesting ones.

The knowledge that genetic programming can accommodate (and in fact excels in) a very complex fitness landscape allowed the use of a complex peer-to-peer network with many parameters in an attempt to achieve greater realism in the simulation. Many of the parameters (such as those in Table B.2) as well as the fitness function for the genetic programming system remained constant throughout all of the scenarios studied. Further work could be done to examine the effects of modifications to these parameters, or to examine the effects of changing the fitness function to push the evolved protocols toward a different optimum.

The grammar of the `CONN` and `RANK` expressions could be modified to include new terminals representing new information gathered from a peer's environment, or to include new operations to be applied to the terminals.

As well, there are many more network scenarios that could be implemented. These include: scenarios with overlapping resource pools instead of the disjoint resource pools used in the scenarios here; sockets that fail with some probability, to develop network protocols that are robust in the face of such unreliability; and transient peers, to develop network protocols that allow new peers to quickly find their appropriate positions in the network. The study of such additional scenarios and the network protocols that are optimal for them are a prime target for future work and would likely result in new insights and the discovery of new methods of self-organization for flooding broadcast networks.

Another direction for future work concerns the heterogeneous use of protocols

within a single network. When generating new flooding broadcast network protocols from the FBMP, the limited processing power available to generate experimental results dictated that it was only practical to apply each generated protocol globally across all peers in the simulated network. It would be more realistic, however, to have each peer run its own genetic programming engine and simulated network, and to be constantly evolving new network protocols that were optimal for that peer's specific context within the network. This approach could be practical in a real-world implementation, where the increased processing power required would be distributed across all peers in the network. Given the current situation of rapidly increasing (and largely underutilized) desktop processing power, such an approach is not unreasonable. This approach would allow a deployed, live peer to optimize and adapt its network protocol to deal with changing network conditions. This strategy would remove the burden of 'pre-processing' the network with a representative traffic sample, but would require each peer to observe the properties of the traffic it was experiencing and to simulate those network conditions in the genetic programming engine it would be running. Building on this, optimal protocols could be passed among peers, ensuring the widespread use of good network protocols and the availability of protocols to peers that have limited processing power to do their own simulations.

Another avenue for future work would be to extend the expressiveness of the FBMP. For example, a 'reduced branching factor' approach could be adopted for message propagation, where peers would individually decide how many of their neighbours they would propagate broadcast messages to. The calculation of this branching factor could be implemented as a third expression beside the FBMP's `CONN` and `RANK` expressions.

A final direction for future work would be to address the routing problem discussed in Section 5.1. This problem concerned reply messages that could not be routed because the chain of peers that they were broadcast on had been broken by a dropped connection. This behaviour imposed an implicit penalty on any network protocol that caused peers to change connections frequently, because the protocol would score a lower number of successful messages due to unroutable messages that were dropped. To get around this penalty, networks could implement a ‘training period’ in which they would be allowed to organize themselves before any fitness measurements were taken.

The extensions and new directions discussed here, together with the results given earlier, show that the approach presented in this thesis is a flexible and effective means for deriving optimal flooding broadcast network protocols for arbitrary network scenarios.

Appendix A

Bandwidth Algorithms

A key algorithm in the simulated network is the method of updating the socket bandwidths when a connection is added or removed. The problem bears some resemblance to flow networks [CLR90], although those algorithms generally deal with directed graphs and do not treat the case where the vertices bear the constraint instead of the edges.

The general problem is, given a graph G and a maximum bandwidth for each vertex in G , what application of bandwidths to edges maximizes the total cumulative bandwidth across the graph subject to each vertex's individual constraint.

In a real network these flows are realized moment-to-moment by the normal greedy bandwidth allocations occurring at each node in the network: if one connection is slow, the extra bandwidth is automatically made available to other connections in an attempt to saturate the link. This does not happen 'for free' in the simulation, so this behaviour must be modeled explicitly.

An online and an offline algorithm were developed, and the offline one is given here. The offline algorithm is slower but provably correct and guaranteed to terminate, and is the one used to obtain the experimental results presented in Chapter 4. The online

algorithm is faster because it updates only the edges and vertices affected by the topology change, but it is significantly more complex and — though it performed correctly in all test cases — its correctness or termination characteristics cannot be guaranteed.

A.1 Offline Algorithm

The offline algorithm is given in Figure A.1. The algorithm accepts a graph G and returns its results by setting the attribute *effective-bandwidth*[e] for each edge e in G .

The algorithm begins by building a min-heap of all the vertices in G , keyed on the attribute *bandwidth-per-edge*[v] (this attribute is calculated on line 14). It proceeds by iteratively removing the vertex that, after dividing its bandwidth evenly over its edges, has the smallest bandwidth allocated to each edge. The edges of the vertex so chosen have their bandwidth set, the neighbouring vertices reallocate their bandwidths accordingly, and the next vertex is drawn from the heap.

```

    CALCULATE-BANDWIDTHS(  $G$  )
1:   for each  $v$  in  $G$ 
       $\triangleright$  Insert  $v$  into min-heap  $H$  keyed on  $bandwidth-per-edge[v]$ 
2:   INSERT-INTO-HEAP(  $v, H$  )
3:   while  $H$  not empty
4:      $v \leftarrow$  EXTRACT-MIN(  $H$  )
5:     while  $edges-to-consider[v]$  not empty
6:        $e \leftarrow$  EXTRACT-FIRST(  $edges-to-consider[v]$  )
7:        $w \leftarrow$  vertex at other end of  $e$  from  $v$ 
       $\triangleright$  Set the bandwidth for edge  $e$ 
8:        $effective-bandwidth[e] \leftarrow bandwidth-per-edge[v]$ 
9:       REMOVE-FROM-LIST(  $e, edges-to-consider[w]$  )
10:      if  $edges-to-consider[w]$  is empty
11:        REMOVE-FROM-HEAP(  $w, H$  )
12:      else
13:         $bandwidth-available[w] = bandwidth-available[w] -$ 
           $bandwidth-per-edge[v]$ 
14:         $bandwidth-per-edge[w] = bandwidth-available[neighbour] /$ 
           $size[edges-to-consider[w]]$ 
       $\triangleright$  Make sure that  $w$ 's position in  $H$  reflects its new key
15:      UPDATE-HEAP-POSITION(  $w, H$  )

```

Figure A.1: The offline bandwidth-calculation algorithm

Appendix B

Simulation Parameters

The parameters used in the genetic programming engine are given in Table B.1. The parameters used in the peer-to-peer simulation are given in Table B.2.

population size	200
maximum tree depth	4
crossover frequency	85%
crossover selection	4-individual tournament
reproduction frequency	5%
reproduction selection	fitness
mutation frequency	10%
mutation selection	4-individual tournament

Table B.1: Genetic programming parameters

join interval	5 s	Number of seconds between each peer's initial request to join the network.
search timeout	5 s	Length of the search window for 'successful' searches. Any successful responses received outside of this window are considered 'timed-out'.
search delay	5 s	Pause after the search window has closed before initiating a new search.
connection update	20 s	Interval at which each peer updates its set of neighbours using the <code>CONN</code> and <code>RANK</code> expressions.
search update	1 s	Interval at which each peer revisits its list of pending searches to determine if any have timed-out.
initial time-to-live	7	Initial time-to-live value for each message. This governs the 'propagation horizon' of the message.
concurrent searches	2	Number of concurrent searches maintained by each peer.
history window	200 s	Length of a peer's immediate history represented in the value returned by each of the terminals in the <code>CONN</code> and <code>RANK</code> expressions.

Table B.2: Peer-to-peer simulation parameters

Bibliography

- [ABK96] David Andre, Forrest Bennett, and John Koza. Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 3–11, Stanford University, CA, USA, 1996. MIT Press.
- [ABKM01] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. The case for resilient overlay networks. In *Proc. of the 8th Annual Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May 2001.
- [Ada] Lada A. Adamic. Zipf, power-laws, and pareto - a ranking tutorial. Available as <http://ginger.hpl.hp.com/shl/papers/ranking/ranking.html>.
- [Ada01] Adaptinet Inc. The adaptinet SDK – programmer’s guide. Available as <http://www.adaptinet.com/ProgrammersGuide.pdf>, May 2001.
- [AH00] Eytan Adar and Bernardo A. Huberman. Free riding on gnutella. *First Monday*, September 2000.
- [AHL01] L. Adamic, B. Huberman, R. Lukose, and A. Puniyani. Search in power law networks. *Physical Review E*, 64(4):46135–46143, 2001.
- [Ash97] Dan Ashlock. GP automata for dividing the dollar. In J. Koza, K. Deb, M. Dorigo, D. Fogel, M. Garzon, H. Iba, and R. Riolo, editors, *Genetic*

Programming 1997: Proceedings of the Second Annual Conference. MIT Press, 1997.

- [BKKA99] Forrest H Bennett III, John R. Koza, Martin A. Keane, and David Andre. Genetic programming: Biologically inspired computation that exhibits creativity in solving non-trivial problems. In *Proceedings of the AISB'99 Symposium on Scientific Creativity*, pages 29–38, Edingburgh, 1999.
- [BMM01] Ozalp Babaoglu, Hein Meling, and Alberto Montresor. Anthill: A framework for the development of agent-based peer-to-peer systems. Technical Report UBLCS-2001-09, University of Bologna, Italy, November 2001.
- [BNC95] Jerry Banks, Barry Nelson, and John Carson. *Discrete-Event System Simulation*. Prentice Hall, 1995.
- [Bor02] John Borland. Ruling bolsters file-traders' prospects. *CNET News.com*, March 2002. Available as <http://news.com.com/2100-1023-870396.html>.
- [Bri95] T. Brisco. DNS support for load balancing. Technical Report RFC 1794, Network Working Group, Rutgers University, April 1995.
- [Cli00] Clip2.com. Gnutella: To the bandwidth barrier and beyond. Available as <http://www.clip2.com/gnutella.html>, November 2000.
- [CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*, chapter 27, pages 579–629. The MIT Press, 1st edition, 1990.
- [CMB00] Y. Chawathe, S. McCanne, and E. Brewer. An architecture for internet content distribution as an infrastructure service. Available as <http://www.cs.berkeley.edu/yatin/papers>, 2000.

- [CMM02] Russ Cox, Athicha Muthitacharoen, and Robert T. Morris. Serving DNS using a peer-to-peer lookup service. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, March 2002.
- [CRZ00] Y. Hua Chu, S. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of SIGMETRICS*, Santa Clara, CA, June 2000.
- [CSWH00] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, 2000.
- [CZH⁺99] Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz. An architecture for a secure service discovery service. In *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networks (MobiCom)*, 1999.
- [Dab01] Frank Dabek. A cooperative file system. Master’s thesis, Massachusetts Institute of Technology, September 2001.
- [DBK⁺01] F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan. Building peer-to-peer systems with chord, a distributed lookup service. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, Schloss Elmau, Germany, May 2001. IEEE Computer Society.
- [DFM00] Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 67–95, December 2000.

- [FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.
- [FIR⁺02] Ian Foster, Adriana Iamnitchi, Matei Ripeanu, Ann Chevernack, Ewa Deelman, Carl Kesselman, Wolfgang Hoschek, Heintz Stockinger, Kurt Stockinger, and Brian Tierney. Giggle: A framework for constructing scalable replica location services. Technical Report GGF4, Global Grid Forum, February 2002.
- [FN97] Dario Floreano and Stefano Nolfi. God save the red queen! competition in co-evolutionary robotics. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 398–406, Stanford University, CA, USA, 1997. Morgan Kaufmann.
- [FV02] Michael J. Freedman and Radek Vingralek. Efficient peer-to-peer lookup based on a distributed trie. In *Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, USA, March 2002.
- [GSG02] P. Krishna Gummadi, Stefan Saroiu, and Steven Gribble. A measurement study of napster and gnutella as examples of peer-to-peer file sharing systems. In *Multimedia Computing and Networking*, January 2002.
- [Gun02] Neil Gunther. Hypernets - good (g)news for gnutella. Available as <http://www.perfdynamics.com/Papers/Gnews.html>, February 2002.
- [GWP96] Frederic Gruau, Darrell Whitley, and Larry Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks.

- In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89, Stanford University, CA, USA, 1996. MIT Press.
- [Hed88] C. Hedrick. Routing information protocol. Technical Report RFC 1058, Network Working Group, Rutgers University, June 1988.
- [HHH⁺02] Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon T. Loo, Scott Shenker, and Ion Stoica. Complex queries in dht-based peer-to-peer networks. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, March 2002.
- [Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor MI, 1975.
- [How96] Brian Howley. Genetic programming of near-minimum-time spacecraft attitude maneuvers. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick R. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 98–106, Cambridge, MA, 1996. MIT Press.
- [HvST97] F. J. Hauck, M. van Steen, and A. S. Tanenbaum. A location service for worldwide distributed objects. *M. Mühlhäuser (ed.) Special Issues in Object-Oriented Programming, dpunkt-Verlag, Heidelberg, (Readings of ECOOP'96 Workshops, Linz, Austria, July 1996)*, pages 384–388, 1997.
- [Iba97] Hitoshi Iba. Multiple-agent learning for a robot navigation task by genetic programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, , and R. L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 195–200, CA, USA, 1997. Stanford University, Morgan Kaufmann.

- [IF01] Adriana Iamnitchi and Ian Foster. On fully decentralized resource discovery in grid environments. In *Proceedings of the International Workshop on Grid Computing*, Denver, Colorado, November 2001. IEEE.
- [IRF02] Adriana Iamnitchi, Matei Ripeanu, and Ian Foster. Locating data in (small-world?) p2p scientific collaborations. In *Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, USA, March 2002.
- [JAB01] Mihaljo A Jovanović, Fred S. Annexstein, and Kenneth A. Berman. Scalability issues in large peer-to-peer networks: A case study of gnutella. *University of Cincinnati Technical Report 2001*, January 2001. Available as <http://www.eecs.uc.edu/~mjovanov/Research/paper.html>.
- [Jov00] Mihaljo A Jovanović. Modeling large-scale peer-to-peer networks and a case study of gnutella. Master's thesis, University of Cincinnati, June 2000.
- [JSBM01] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. DNS performance and the effectiveness of caching. In *The ACM SIGCOMM Internet Measurement Workshop*, San Francisco, California, November 2001.
- [KABK96] John R. Koza, David Andre, Forrest H. Bennett III, and Martin A. Keane. Use of automatically defined functions and architecture-altering operations in automated circuit synthesis using genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 132–149, Stanford University, CA, USA, 1996. MIT Press.

- [KK00] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Mobile Computing and Networking*, pages 243–254, 2000.
- [KLL⁺97] David R. Karger, Eric Lehman, Frank Thomson Leighton, Rina Panigrahy, Matthew S. Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 654–663, May 1997.
- [KMG00] A. Kermarrec, L. Massoulie, and A. Ganesh. Reliable probabilistic communication in largescale information dissemination systems. Technical Report MSRTR-2000-105, Microsoft Research, Cambridge, 2000.
- [Koz92] John R. Koza. *Genetic Programming: On the programming of computers by means of natural selection*. MIT Press, 1992.
- [Koz99] John R. Koza. Twenty-five instances where genetic programming has produced human-competitive results. Available as <http://www.genetic-programming.com/humancompetitive.html>, 1999.
- [LCC⁺01] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. Preprint, 2001.
- [Len97] Terje Lensberg. A genetic programming experiment on investment behavior under knightian uncertainty. In J. Koza, K. Deb, M. Dorigo, D. Fogel, M. Garzon, H. Iba, and R. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*. MIT Press, 1997.
- [LRS02] Qin Lv, Sylvia Ratnasamy, and Scott Shenker. Can heterogeneity make gnutella scalable? In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

- [Mah01] R. Mahajan. How Akamai works. Available as <http://www.cs.washington.edu/homes/ratul/akamai.html>, 2001.
- [MD88] Paul V. Mockapetris and Kevin J. Dunlap. Development of the domain name system. In *SIGCOMM*, pages 123–133, 1988.
- [Mil67] Stanley Milgram. The small world problem. *Psychology Today*, pages 60–67, May 1967.
- [Mon01] Alberto Montresor. Anthill: a framework for the design and analysis of peer-to-peer systems. In *Proceedings of the 4th European Research Seminar on Advances in Distributed Systems*, Bertinoro, Italy, May 2001.
- [MRR80] J. McQuillan, I. Richer, and E. Rosen. The new routing algorithm for the arpanet. *IEEE Transactions on Communications*, Com-28(5):711–719, May 1980.
- [Nap] Napster Inc. Napster homepage. <http://www.napster.com/>.
- [New00] M.E.J Newman. Models of the small world: A review. *Journal of Statistical Physics*, 101:819–841, 2000.
- [NW] M. Newman and D. Watts. Renormalization group analysis of the small-world network model.
- [NWS02] M.E.J. Newman, D.J. Watts, and S.H. Strogatz. Random graph models of social networks. In *Proc. Natl. Acad. Sci. USA*, volume 99, pages 2566–2572, 2002.
- [Ora01] Andy Oram, editor. *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*. O’Reilly, Sebastopol, CA, March 2001.

- [Pec02] Martin Peck. Decentralized resource discovery in large peer-based networks. Available as <http://cubicmetercrystal.com/alpine/discovery.html>, 2002.
- [PRR97] C. Plaxton, R. Rajaram, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, June 1997.
- [pts02] ptsc@nym.alias.net. The Church of Scientology’s supremacy over the search term ‘scientology’ on Google. Available as <http://www.operatingthetan.com/google/>, February 2002.
- [PZ98] Bill Punch and Douglas Zongker. The lil-gp genetic programming system. Available as <http://garage.cps.msu.edu/software/lil-gp/lilgpindex.html>, 1998.
- [RD01] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, November 2001.
- [RE96] T.W. Ryu and C.F. Eick. Masson: Discovering commonalities of a set of objects using genetic programming. In *Proceedings of the Genetic Programming 1996 Conference*, CA, July 1996. Stanford University.
- [RFH⁺00] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. Technical Report TR-00-010, University of California at Berkeley and ACIRI and Tahoe Networks, Berkeley, CA, 2000.

- [RFI02] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, 6(1), January/February 2002.
- [RHKS01] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-level multicast using content-addressable networks. In *NGC*, 2001.
- [RHKS02] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Topologically-aware overlay construction and server selection. In *Info-com*, 2002.
- [Rip01] Matei R. Ripeanu. Peer-to-peer architecture case study: Gnutella network. Technical Report TR-2001-26, University of Chicago, July 2001.
- [Rit01] Jordan Ritter. Why gnutella can't scale. no, really. Available as <http://www.darkridge.com/jpr5/doc/gnutella.html>, February 2001.
- [RSS97] Jonathan Rosenberg, Henning Schulzrinne, and Bernhard Suter. Wide area network service location. Available as <http://www.globecom.net/ietf/draft/draft-ietf-srvloc-wasrv-01.html>, November 1997.
- [SAR⁺02] Ion Stoica, Dan Adkins, Sylvia Ratnasamy, Scott Shenker, Sonesh Surana, and Shelley Zhuang. Internet indirection infrastructure. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.
- [SBK01] Bujor Silaghi, Samrat Bhattacharjee, and Pete Keleher. Query routing in the terradir distributed directory. Submitted for publication, 2001.
- [SMK⁺01] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications.

- Technical Report TR-819, Massachusetts Institute of Technology, March 2001.
- [Sri01] Kunwadee Sripanidkulchai. The popularity of gnutella queries and its implications on scalability. Available as <http://www-2.cs.cmu.edu/kunwadee/research/p2p/paper.html>, March 2001.
 - [Suna] Sun Microsystems. Jini technology architectural overview white paper. Available as <http://www.sun.com/jini/whitepapers/architecture.html>.
 - [Sunb] Sun Microsystems. Project JXTA: A technology overview. White paper available at http://www.jxta.org/project/www/white_papers.html.
 - [Sunc] Sun Microsystems. JXTA v1.0 protocols specification. White paper available as <http://spec.jxta.org/v1.0/docbook/JXTAProtocols.html>.
 - [Tho96] Adrian Thompson. Silicon evolution. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 444–452, Stanford University, CA, USA, 1996. MIT Press.
 - [Tru01] Kelly Truelove. The juxtanet. *O'Reilly OpenP2P.com*, April 2001. Available as <http://www.openp2p.com/pub/a/p2p/2001/04/25/juxtanet.html>.
 - [Wal99] Toby Walsh. Search in a small world. In *IJCAI*, pages 1172–1177, 1999.
 - [Wat99] Duncan Watts. *Small Worlds*. Princeton University Press, Princeton, 1999.
 - [WS98] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393:440–442, June 1998.

- [ZKJ00] Ben Y. Zhao, John Kubiawicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, U. C. Berkeley, April 2000.
- [ZZJ⁺01] S. Q. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiawicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the Eleventh International Workshop on Network and OS Support for Digital Audio and Video*, New York, July 2001. ACM.